
图灵电子书
编程小白的第1本
Python入门书
侯爵

写在前面：你需要这本书的原因

有没有哪一个瞬间，让你想要放弃学习编程?

在我决心开始学编程的时候，我为自己制定了一个每天编程1小时的计划，那时候工作很忙，我只能等到晚上9点，同事都下班之后，独自留在办公室编程。在翻遍了我能找到的几十本国内外的 Python 编程教程之后，我还是似懂非懂。那些教程里面到处都是抽象的概念、复杂的逻辑，对于专业开发者这些再平常不过，而对于我这样一个学设计出身的编程小白，没有被视觉化的东西是无法被理解的。

而且，这些书大多着重于一步步构建一个完整体系，但事实上，现实生活中没有哪个技能是这么习得的。难道要练习1年切菜才能给自己做一顿饭么？难道要到体校学习3年才能开始晨跑么？难道要苦练5年基本功才能开始拿起吉他弹第1首曲子么?

做任何事情一定有在短期内简单可行的方法。学习不应该是苦差事，而应该是快乐的，重要的是找到适合自己的学习方法。

既然笨办法不能让我学会 Python，那么我决定用一种聪明方法来学，为自己创造学习的捷径。这种高效学习法的核心在于：

1、**精简**：学习最核心的关键知识；

2、**理解**：运用类比、视觉化的方法来理解这些核心知识；

3、**实践**：构建自己的知识体系之后，再通过实践去逐渐完善知识体系。

实际上，如果你听说过《如何高效学习》中的整体学习法，你会发现它和我的高效学习法很相似，作者斯科特·杨用一年的时间学完了麻省理工四年的课程。既然这种高效学习法可以用来学习经济学、数学、物理，那么当然也可以用来学编程。



运用了高效学习法之后，我的编程学习速度突飞猛进，不到一个月时间就完成了一个爬虫加上 Django 的网站。为了追求最大程度的精简，我借用了许多强大的库和框架，这让我不需要重新发明轮子，而是专注于最核心的功能。在一次项目合作中，我惊讶的发现，我用70行代码实现的功能，一名工程师竟然用了800行代码来实现。在运动场上，第一名和最后一名的差距也许只有几十秒。然而在编程这个竞技场上，**不同方法的效率竟然会有10倍的差距，这也正是追求高效学习的价值所在。**

为了能让更多的编程小白轻松地入门编程，我把高效学习法结合 Python 中的核心知识，写成了这本书。随意翻上几页，你就会发现这本书和其他编程书的不同，其中有大量的视觉化场景帮你理解概念，穿插了若干有趣的小项目，**最重要的是，这本书是为零基础小白而设计的。**

考虑到很多书都标榜是针对零基础读者的，所以有必要说明一下这一本有哪些独到之处。

我不写字典式的知识体系，有些编程书像字典一样，各种细枝末节的知识都涵盖在内。但我认为，正如不应该让小孩拿着字典学汉语，也不应该让初学者拿着这样的厚书学编程。在汉语中，掌握常见的1500-2000个字就能看懂80%的文字。在编程中，同样有着最核心的关键知识。**先用这些关键的知识构建你的知识体系会让学习效率加速，这是比一开始就钻到各种细枝末节里面更好的学习方式。**这是精简的意义。

我不去对比各种语言的特点，许多程序员背景的作者喜欢去对比 Python 和其他语言有什么异同，或者试图让你通过理解 C 语言从而理解 Python，但我不会这么做。我知道对于大多数读者，Python 很可能是将要学习的第一门编程语言，所以我不会用一个陌生概念讲解另一个陌生概念，反过来，**我会运用类比和视觉化的方法讲解 Python 中的抽象概念，把复杂的东西简单的讲清楚。**这是理解的捷径。

我不追求让你达到精通的程度，事实上我也很怀疑有哪本书能真正做到21天从入门到精通。精通一门语言，需要在实际项目开发中踩过许多坑，需要熟悉计算机运作的底层原理。我是一名实用主义的开发者，我相信你也一样，学习编程是为了真正做出点东西来，也许你想爬取大量的数据和信息，方便用来分析与决策。也许你想快速搭建一个网站，展示自己的产品。也许你对量化交易感兴趣，想试着把自己的投资策略程序化。对于实用主义的开发者来说，更应该追求的是“达成”而不是“精通”。**先掌握项目所需的最少必要知识，然后把热情和精力投入到搭建真实项目中，而不是死磕半年的基础知识，直到把所有兴趣都耗竭了也没做出来什么像样的东西。**在实践过程中，你自然会逐渐完善知识体系。在这本书里面，会穿插一些真实项目的片段，让你知道学了这个基础知识能用在哪，并且完成一些小型项目。这是让你最有成就感的实践。

说了这么多，就是为了让你能放下疑虑。这不是一本让你中途放弃的编程书，这是一本黏着你看完的编程书。大多数读者都能在一周内读完，其中有35岁才开始学编程的中年男子、有工作非常忙碌的女性创业者、还有对编程感兴趣的高中生。所以，相信你也可以跟着这本书一起从零到一。

放轻松，如果你准备好了，那就翻开下一页吧。



作者介绍

麻瓜编程创始人。网易云课堂上最畅销的课程《Python 实战》系列课程讲师，目前已有超过4万名学员。

设计专业背景，拥有设计与编程跨界思维，善于找到学习技能的最佳路径，擅长把复杂的东西简单的讲清楚。

初学编程时，发现市面上很难找到适合小白的学习资料，于是开始用生动易懂、视觉化的方式来写这本教程。

目录

第一章

为什么选择 Python？

Why Python?

那些最好的程序员不是为了得到更高的薪水或者得到公众的仰慕而编程，他们只是觉得这是一件有趣的事情。

——Linux 之父 Linus Torvalds

作为一个实用主义的学习者，最关心的问题一定是「我为什么要选择学 Python，学会之后我可以用来做什么？」

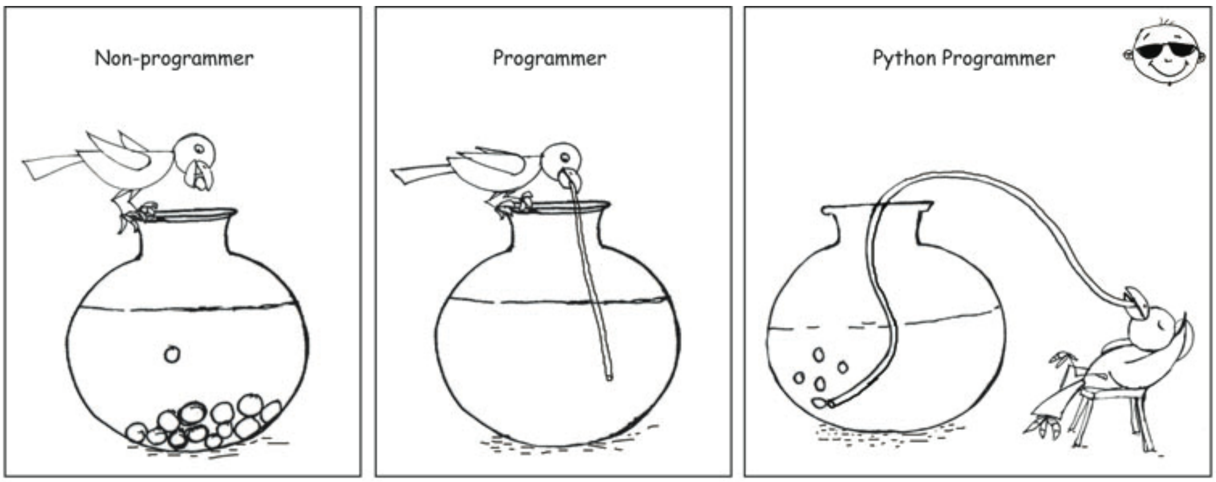
首先，对于初学者来说，比起其他编程语言，Python 更容易上手。

Python 的设计哲学是优雅、明确、简单。在官方的 The Zen of Python（《Python 之禅》）中，有这样一句话，

> There should be one-- and preferably only one --obvious way to do it.

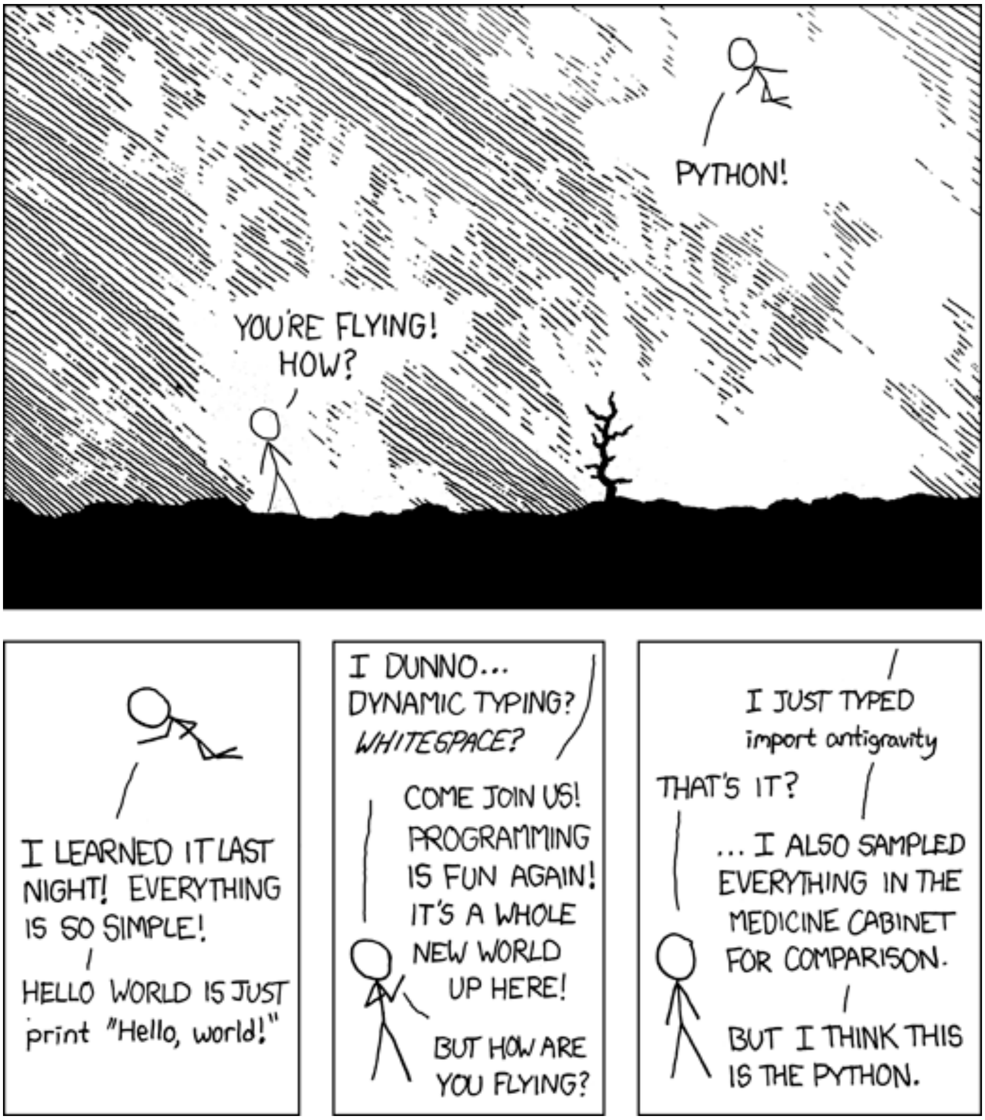
Python 追求的是找到最好的解决方案。相比之下，其他语言追求的是多种解决方案。

如果你试着读一段写的不错的 Python 代码，会发现像是在读英语一样。这也是 Python 的最大优点，它使你能够专注于解决问题而不是去搞明白语言本身。



注：漫画《口渴的 Python 开发者》，形容了 Python 开发者是多么轻松，来自 Pycot 网站。

其次，Python 功能强大，很多你本来应该操心的事情，Python 都替你考虑到了。当你用 Python 语言编写程序的时候，你不需要考虑如何管理你的程序使用的内存之类的底层细节。并且，Python 有很丰富的库，其中有官方的，也有第三方开发的，你想做的功能模块很有可能已经有人写好了，你只需要调用，不需要重新发明轮子。这就像是拥有了智能手机，可以任意安装需要的 app。



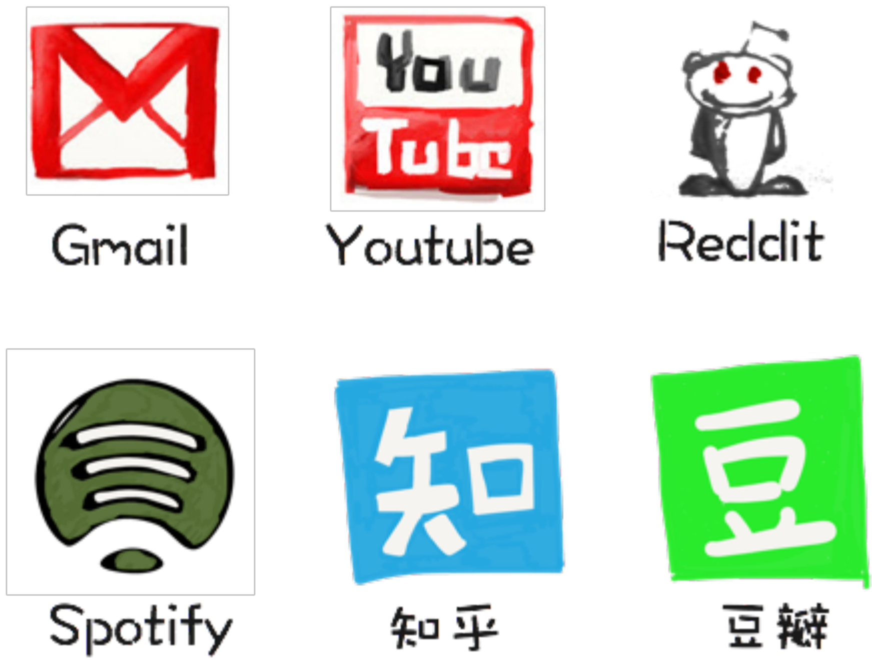
注：漫画 Python，作者xkcd

这幅漫画形容了 Python 的库有多强大，导入一个反重力库就可以飞起来了。

第三，Python 能做的事情有许多。

在职场中，使用 Python 工作的主要是这样几类人：

- 网站后端程序员：使用 Python 搭建网站、后台服务会比较容易维护，当需要增加新功能，用 Python 可以比较容易的实现。不少知名网站都使用了 Python 开发，比如：

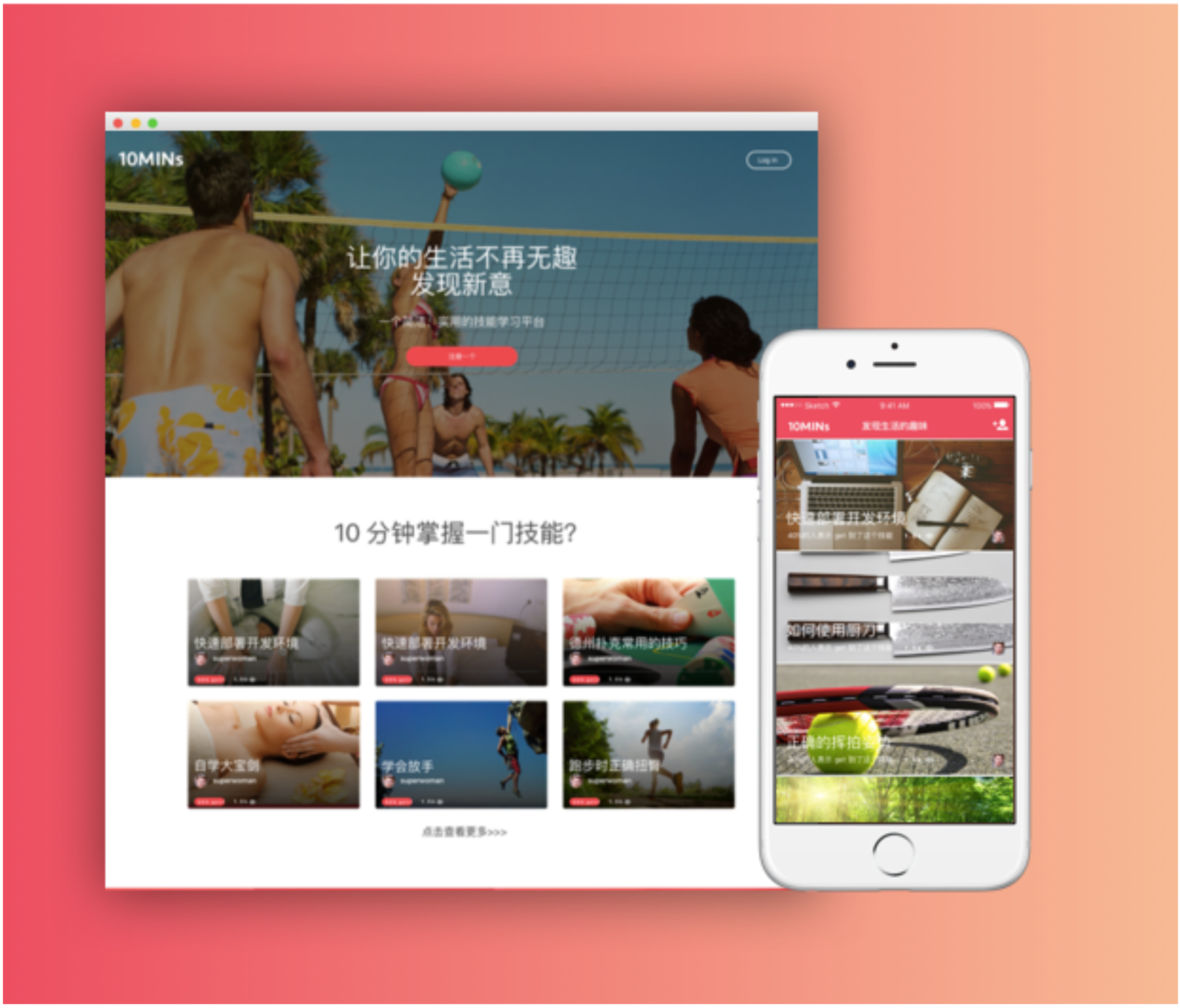


- 自动化运维：越来越多的运维开始倾向于自动化，批量处理大量的运维任务。Python 在系统管理上的优势在于强大的开发能力和完整的工具链。
- 数据分析师：Python 能快速开发的特性可以让你迅速验证你的想法，而不是把时间浪费在程序本身上，并且有丰富的第三方库的支持，也能帮你节省时间。
- 游戏开发者：一般是作为游戏脚本内嵌在游戏中，这样做的好处是即可以利用游戏引擎的高性能，又可以受益于脚本化开发的优点。只需要修改脚本内容就可以调整游戏内容，不需要重新编译游戏，特别方便。
- 自动化测试：对于测试来说，要掌握 Script 的特性，会在设计脚本中，有更好的效果。Python 是目前比较流行的 Script。

如果你是一名业余开发者，只是想在资源少的情况下快速做出自己想要的东西、自动化的解决生活中的问题，那么 Python 可以帮你做到这几类事情：

- 网站的开发

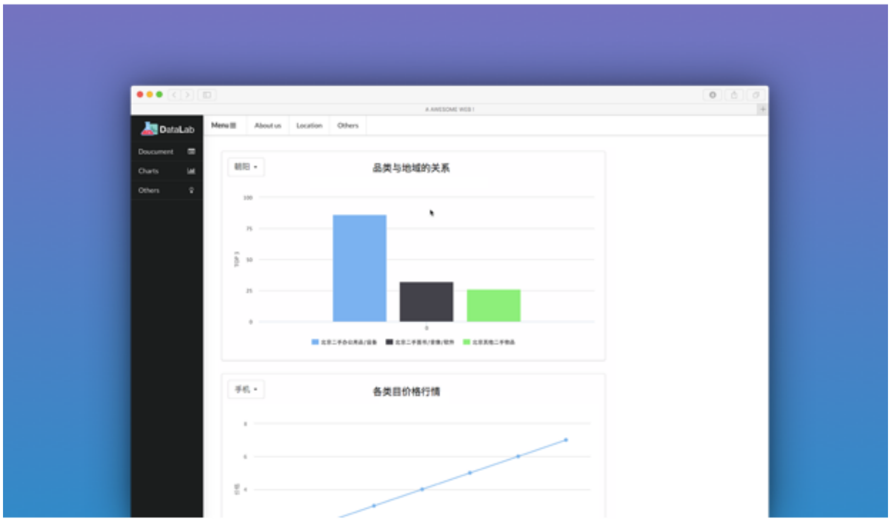
借助功能丰富的框架 django,flask,丰富的设计模板 bootstrap ,你可以快速搭建自己的网站，还可以做到移动端自适应。



注：Python 全栈实战课程的项目：十分钟短视频平台

- 用爬虫爬取或处理大量信息

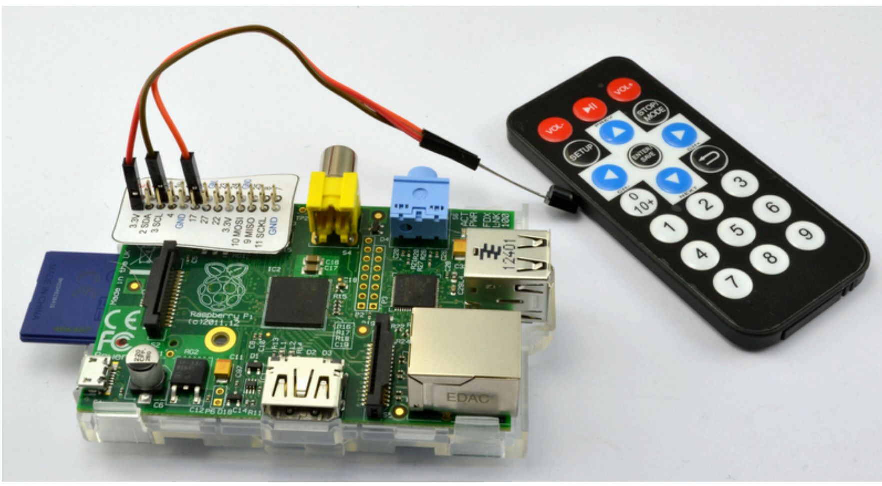
当你需要获取大批量数据或是批量处理的时候，Python 爬虫可以快速做到这些，从而节省你的重复劳动时间。比如：微博私信机器人、批量下载美剧、运行投资策略、刷折扣机票、爬合适房源、系统管理员的脚本任务等等。



注：Python 爬虫实战课程的项目：二手行情网站

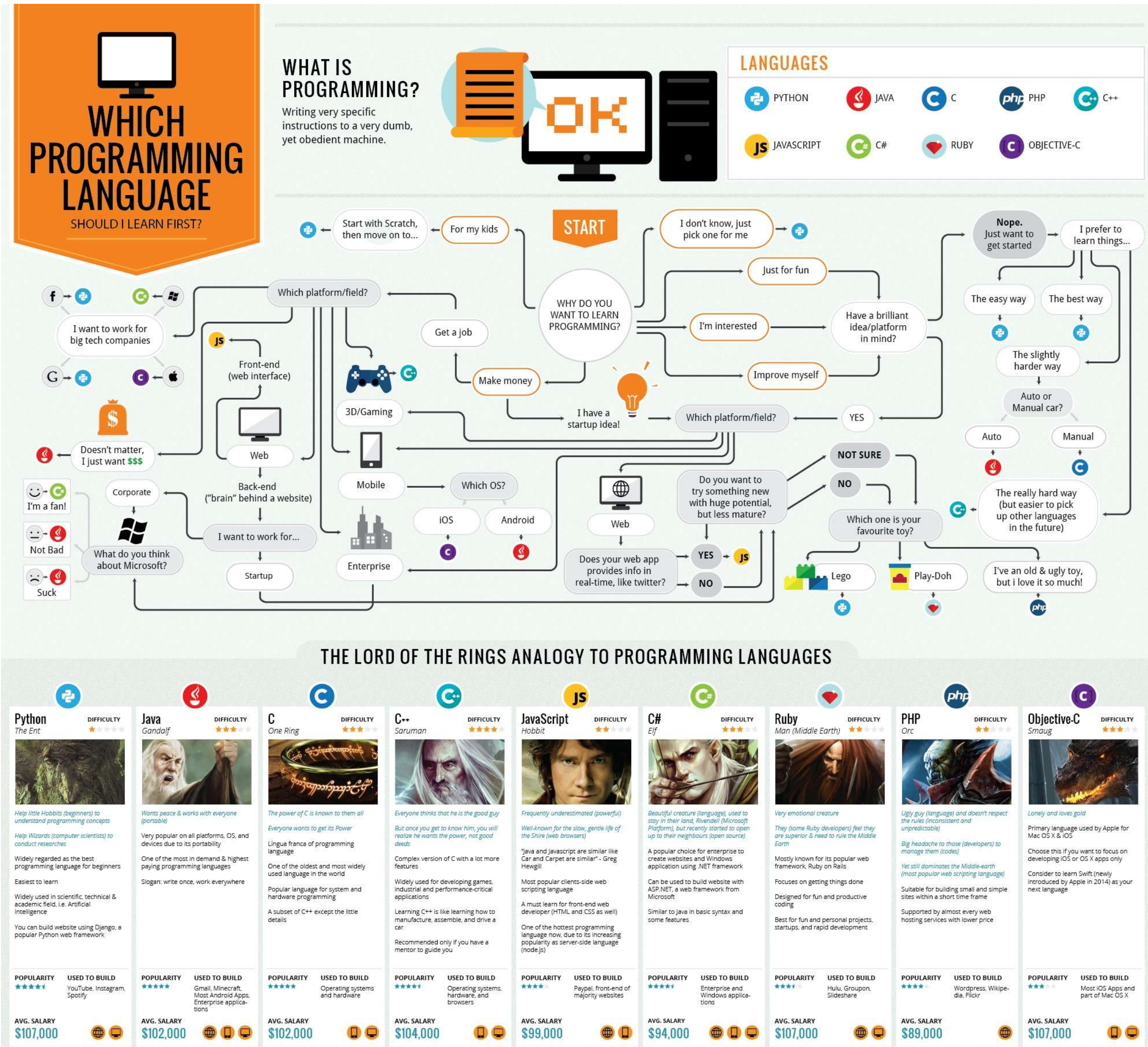
- 再包装其他语言的程序

Python 又叫做胶水语言，因为它可以用混合编译的方式使用c/c++/java等等语言的库。 另外，树莓派作为微型电脑，也使用了 Python 作为主要开发语言。



注：用红外线遥控器控制树莓派，作者八宝粥

最后，附一张选择编程语言的小测试，你可以根据你的需要，选择学习哪种语言。



注：The lord of the rings，作者 carlcheo

第二章

现在就开始

安装 Python 环境

在你开始学习 Python 之前最重要的是——对，你要安装 Python 环境。许多初学者会纠结应该选择 2.x 版本还是 3.x 版本的问题，在我看来，世界变化的速度在变得更快，语言的更新速度速度亦然。没有什么理由让我们只停留在过去而不往前看。对于越来越普及、同时拥有诸多炫酷新特性的 Python 3.x，我们真的没有什么理由的拒绝它。如果你理解了 life is short，you need Python 的苦衷，就更应该去选择这种「面向未来」的开发模式。

所以，我们的教材将以最新的 Python 3.x 版本为基础，请确保电脑上有对应版本。

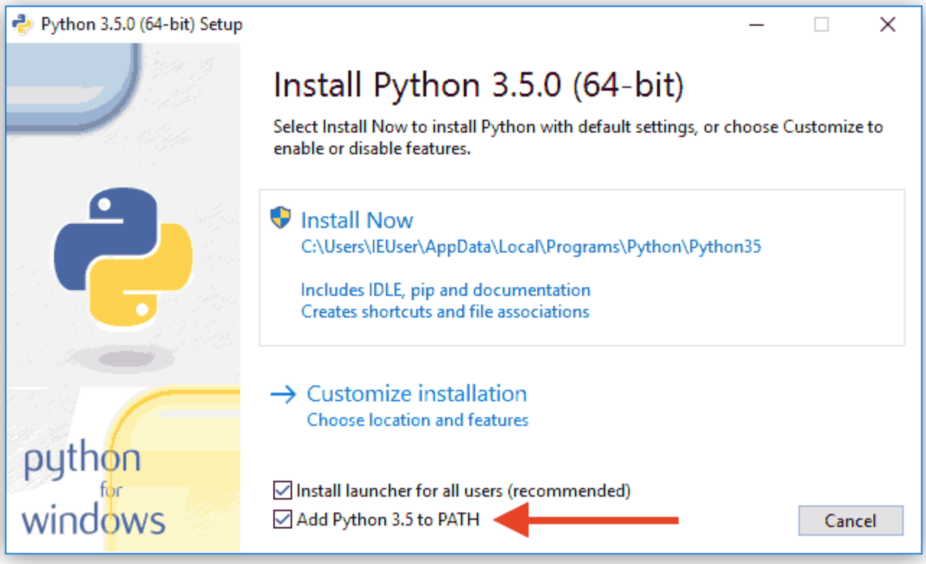
在 Windows 上安装 Python

第一步

根据你的 Windows 版本（64位还是32位）从 Python 的官方网站下载对应的 Python 3.5， 另外，Windows 8.1 需要选择 Python 3.4,地址如下：

- Python 3.5 64位安装程序 https://www.Python.org/ftp/Python/3.5.0/Python-3.5.0-amd64.exe
- Python 3.5 32位安装程序 https://www.Python.org/ftp/Python/3.5.0/Python-3.5.0.exe
- Python 3.4 64位安装程序 https://www.Python.org/ftp/Python/3.4.3/Python-3.4.3.amd64.msi
- Python 3.4 32位安装程序 https://www.Python.org/ftp/Python/3.4.3/Python-3.4.3.msi
- 网速慢的同学请移步国内镜像 http://pan.baidu.com/s/1bnmdlZx

然后，运行下载的EXE安装包：

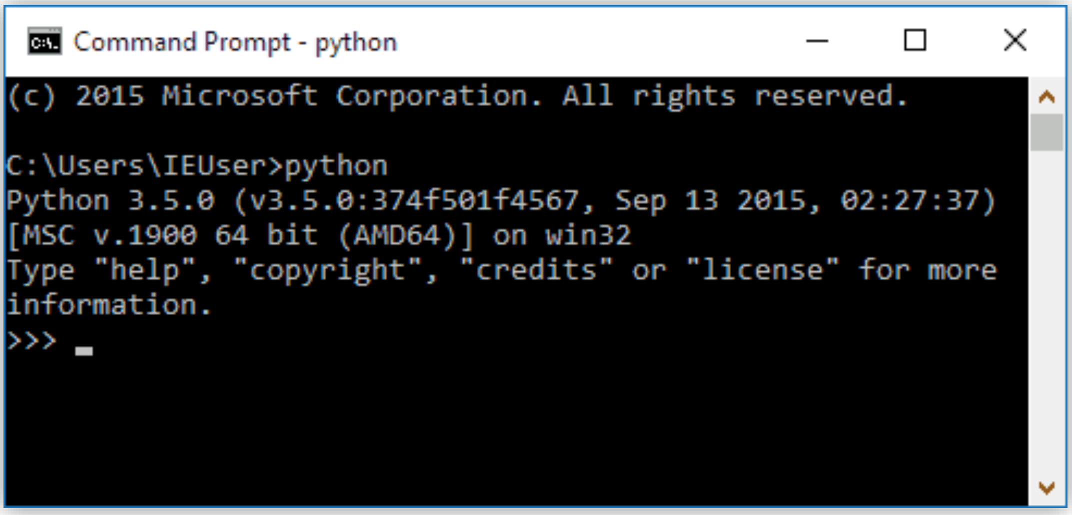


特别要注意勾上 Add Python 3.5 to PATH，然后点“Install Now”即可完成安装。默认会安装到 C:\Python35 目录下。

第二步

打开命令提示符窗口（方法是点击“开始”–“运行”–输入：“cmd”），敲入Python后，会出现两种情况：

- 情况一：

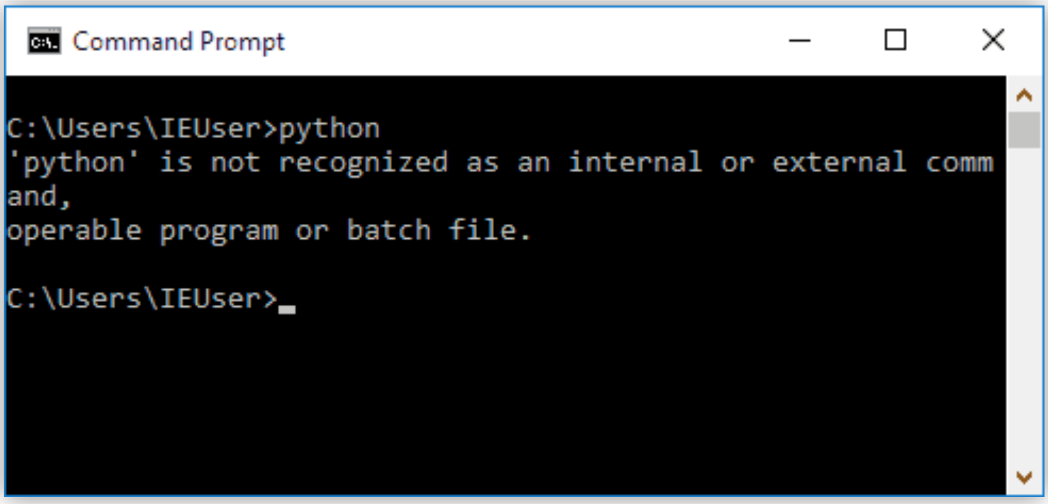


看到上面的画面，就说明 Python 安装成功！

你看到提示符>>>就表示我们已经在 Python 交互式环境中了，可以输入任何 Python 代码，回车后会立刻得到执行结果。现在，输入exit()并回车，就可以退出Python 交互式环境（直接关掉命令行窗口，或者使用快捷键“Ctrl+C”也可以）。

- 情况二：得到一个错误：

  ‘Python’不是内部或外部命令，也不是可运行的程序或批处理文件。



这是因为Windows会根据一个Path的环境变量设定的路径去查找Python.exe，如果没找到，就会报错。如果在安装时漏掉了勾选Add Python 3.5 to PATH，那就要手动把Python.exe所在的路径添加到Path中。

如果你不知道怎么修改环境变量，建议把Python安装程序重新运行一遍，务必记得勾上Add Python 3.5 to PATH。

在 Mac 上安装 Python

如果你正在使用 Mac，系统是OS X 10.8~10.10，那么系统自带的Python版本是 2.7，需要安装最新的 Python 3.5。

第一步：

- 方法一：从 Python 官网下载 Python 3.5 安装程序
  - 链接：https://www.Python.org/ftp/Python/3.5.0/Python-3.5.0-macosx10.6.pkg
  - 网速慢的同学请移步国内镜像：http://pan.baidu.com/s/1sjqOkFF

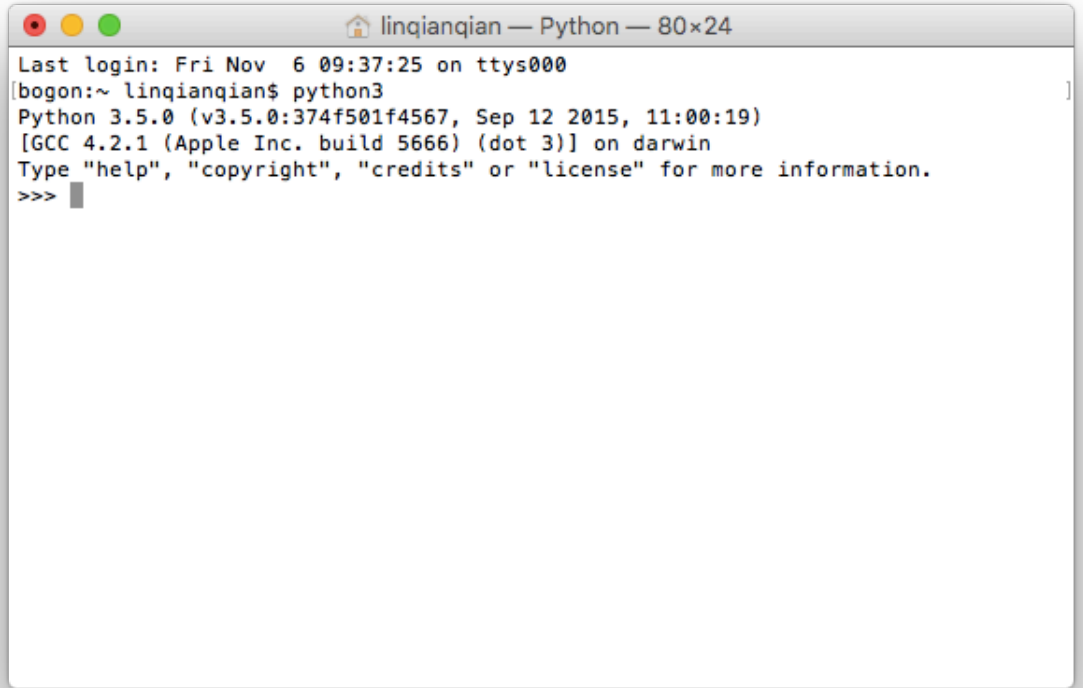
Mac 的安装比 Windows 要简单，只需要一直点击继续就可以安装成功了。



- 方法二：如果安装了 Homebrew，直接通过命令brew install Python3安装即可。

第二步

如果不放心，可以再检查一下。操作方法是打开终端，输入Python3（不是输入 Python 3,也不是 Python）



得到这样的结果，就说明安装成功了。

在 Linux 上安装 Python

一个好消息是，大多数 Linux 系统都内置了 Python 环境，比如 Ubuntu 从13.04 版本之后，已经内置了 Python 2和 Python 3两个环境，完全够用，你不需要再折腾安装了。

如果你想检查一下 Python 版本，打开终端，输入：

```
python3 --version
```

就可以查看 Python 3 是什么版本的了。

如果你需要安装某个特定版本的 Python，在终端输入这一行就可以：

```
sudo apt-get install python3.5
```

其中的3.5可以换成你需要的版本，目前 Python 最新是 3.5 版。

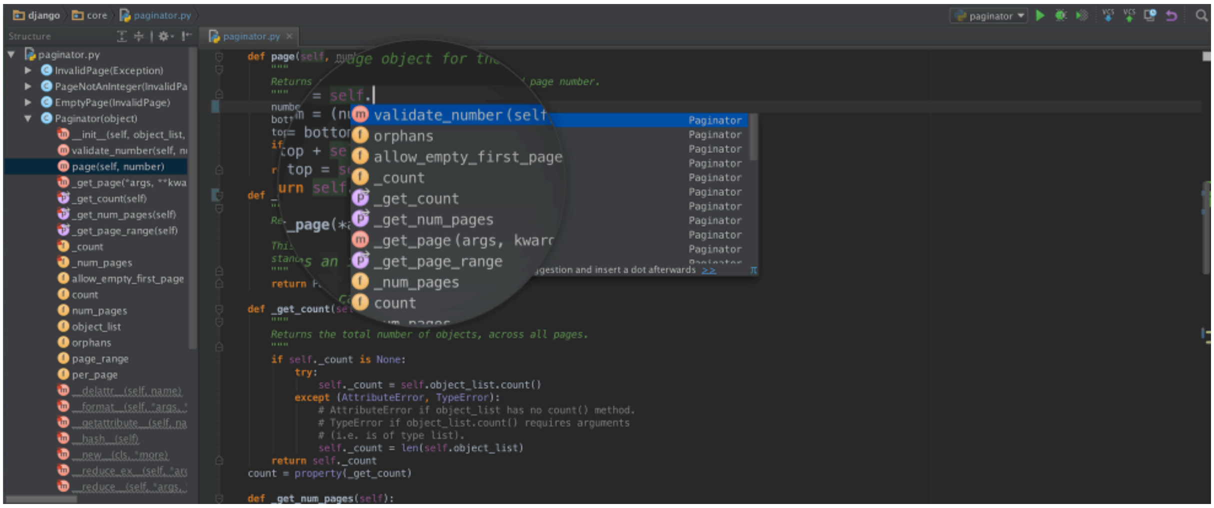
使用 IDE 工具

安装好环境之后，还需要配置一个程序员专属工具。正如设计师使用 Photoshop 做图、产品经理使用 Axure 做原型，程序员也有编程的工具，叫做：IDE。

在这里推荐公认最智能最好用的 Python IDE，叫做 PyCharm，同时支持 windows 和 mac 用户，本教程使用的版本是目前最新的3.4版本。

官网下载链接是：https://www.jetbrains.com/PyCharm/

社区版是免费的，专业版是付费的。对于初学者来说，两者的差异微乎其微，使用社区版就够用了。



到这里，Python 开发的环境和工具就搭建好了，由于 PyCharm 的使用极其简单，几乎不需要学习额外的教程，我们可以开始安心编程了。

如果你是第一次上手编程，可能会对 IDE 感到很陌生，甚至不知道怎样创建一个新文件。在这里推荐一些容易上手的 PyCharm 学习视频：

- 快速上手的中文视频
  http://v.youku.com/v_show/id_XODMyMzM1NzQ4.html

  简单介绍了如何安装、如何创建文件、如何设置皮肤。新手先掌握这些就够用了。
- PyCharm 官方的快速上手视频
  https://www.jetbrains.com/pycharm/documentation/

  第一节视频就让你快速掌握这个工具的基本使用方法，如果你想继续深入了解，可以继续看后面8节短视频，每个在3-5分钟，全面的介绍了如何更有效率的使用 PyCharm。

- 如何高效使用 PyCharm 的系列文档
  http://pedrokroger.net/getting-started-pycharm-python-ide/
  图文并茂的介绍了许多高效的技巧，比如快捷键设置等等，可以在上手之后持续学习。

可能有些同学会有疑问，下面解答一下。

为什么不需要安装 Python 解释器?

> 因为在 Python 官方网站下载了 Python 3.5之后，就自带了官方版本的解释器，所以不需要再次安装。

为什么不使用文本编辑器，比如 Sublime ?

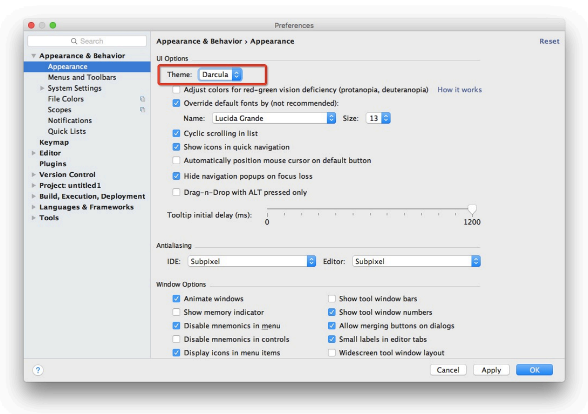
> 因为文本编辑器是相对轻量级的，和 IDE 相比功能太弱了，尤其在 debug 的时候会遇到很多问题。

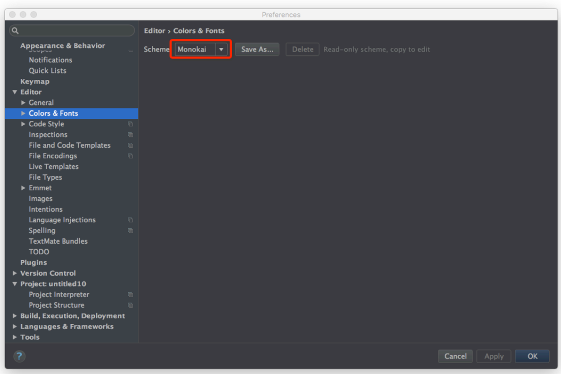
能不能不安装 IDE，直接在命令行或者终端里编程?

> 可以。但是在命令行中保存完整代码很麻烦，最重要的是编辑器是交互式的，不小心手滑写错的代码无法修改，要重新敲一遍。珍惜时间，善用工具。

教程里使用的是什么主题和颜色?

> 皮肤主题 theme 使用的是 darcula，字体着色使用的是 Monikai。





第三章

基础中的基础
变量与字符串

开始学习编程

初学者经常会遇到的困惑是，看书上或是听课都懂，但还是不明白要怎么编程。这是因为缺乏足够多的实践。

正如我们在婴儿时期学习说话的时候，最初是模仿父母的发音，逐渐才能学会表达自己的想法。学习编程也是一样，在你阅读这本教程的时候，需要模仿着示例敲一遍代码，不要怕麻烦、不要嫌简单，当你动手敲代码的时候，就会发现很多眼睛会忽略的细节：小到中文标点还是英文标点、大到语句之间的逻辑关系。当然，在你发现亲手写出的程序运行成功之后，你也会感受到无比的喜悦，你能用程序计算数学题了！你能实现小功能了！我会带着你循序渐进地完成一个个实践，直到你有能力脱离模仿、开始创造。

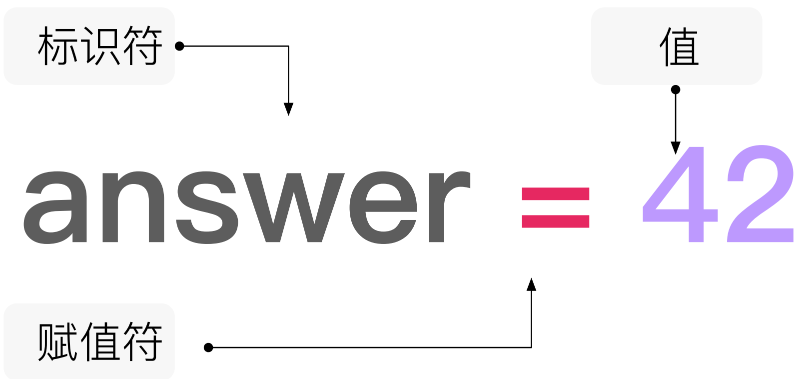
所以，你需要在阅读时打开IDE,边看边敲代码。如果安装环境这一步遇到了问题，请回到上一章阅读。

如果你准备好了，跟我走吧。

变量

简单地说，变量就是编程中最基本的存储单位，变量会暂时性地储存你放进去的东西。

《银河系漫游指南》里面说“生命、宇宙以及任何事情的终极答案是42”，如果用编程语言来表达的话，就是如下等式，一个叫做“answer”的变量被赋值为42。正如每个人都有姓名一样，变量的名字叫做标识符。



现在我们来试着给变量赋值。为了最简单的完成这一步，Windows用户请打开命令行输入 Python 并回车，Mac 用户打开终端输入 Python3 并回车，然后输入：

```
a = 12
```

这样就完成了 a 的赋值，试着回车换行并输入 `a` ，再回车之后，你会看到赋值的结果是12。

需要注意的是，Python 对大小写敏感，也就是说 “a” 和 “A” 会是两个不同的变量，而不是同一个。

这样，你就学会给变量起名字了，并且他们随叫随到。

print()

打印是 Python 中最常用的功能，顾名思义，我们现在就简单把 print() 这个功能理解为展示打印的结果。使用方法是把你要打印查看结果的对象塞进括号中，这样就可以了。（如果你的 print 不用括号也能使用，请检查你的 Python 版本，为了方便快速理解编程概念和少走弯路，后面的所有例子都会用 Python 3.x 实现。）



如果你使用命令行或终端直接输入 print(a)，你会得到下图的结果。这是因为你漏掉了变量的赋值，Python 是无法打印不存在的对象的。

```
>>> print(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
>>>
```

在今后的学习，中我们还有很多很多的东西要进行“打印”，我们需要知道要打印的东西是什么。即便变量是最容易理解的基础知识，也不要因为简单就随意命名，一定要保持 Python 的可读性。

看看下面这段代码，即便你现在不知道其中一些细节，但是读了一遍之后，你也能大概猜到这段代码做了什么事情吧?

```
file = open('/Users/yourname/Desktop/file.txt','w')
file.write('hello world!')
```

这是你敲的第一段代码，所以在这里多说几句。首先需要注意语法问题，使用英文标点符号、大小写不要出错、空格不能少。其次要注意文件路径问题，你的桌面上不需要有 file.txt这个文件，但你需要知道你的电脑上桌面文件的路径是什么，然后把`/Users/yourname/Desktop/`替换掉。查看文件路径的方法是，windows 用户用资源管理器打开桌面上的一个文件，查看路径。Mac 用户打开终端 terminal，然后把桌面上的某个文件拖拽进去就可以查看到路径。

这段代码打开了桌面上的file.txt 文件，并写入了 Hello World！w代表着如果桌面上有file.txt这个文件就直接写入 hello world,如果没有file.txt这个文件就创建一个这样的文件。

互联网上有着诸多的代码和教程，但如果你没能一眼看懂这段代码是什么意思，其中有一多半是因为变量命名不清楚造成的。因此在随后的教程中，哪怕很啰嗦，我也会使用清晰的命名方式，从而来保证即便是没有计算机基础的人，也能够理解代码。

要保持良好的命名习惯应该尽量使用英文命名，学编程的同时还能背单词，岂不一举两得，过一阵子你就会发现英文教程也会阅读得很顺畅。

在这里先了解这么多，更深入的会在之后介绍。

扩展阅读：

- 驼峰式命名法 https://zh.wikipedia.org/wiki/駝峰式大小寫
- 帕斯卡命名法 https://zh.wikipedia.org/wiki/帕斯卡命名法

字符串

字符串是什么

在上面我们已经初步接触到了字符串，很简单地说，字符串就是……..

“任何在这双引号之间的文字”

或者

‘单引号其实和双引号完全一样’

再或者

‘‘‘三个引号被用于过于长段的文字或者是说明，只要三引号不完你就可以随意换行写下文字’’’

字符串的基本用法

现在我们来试着了解一些字符串的基本用法——合并。请在你的 IDE（也就是前面推荐的 PyCharm） 中输入如下代码，在 IDE 中代码并不能自动运行，所以我们需要手动点击运行，方法是点击右键，选择“Run‘文件名’”来运行代码。

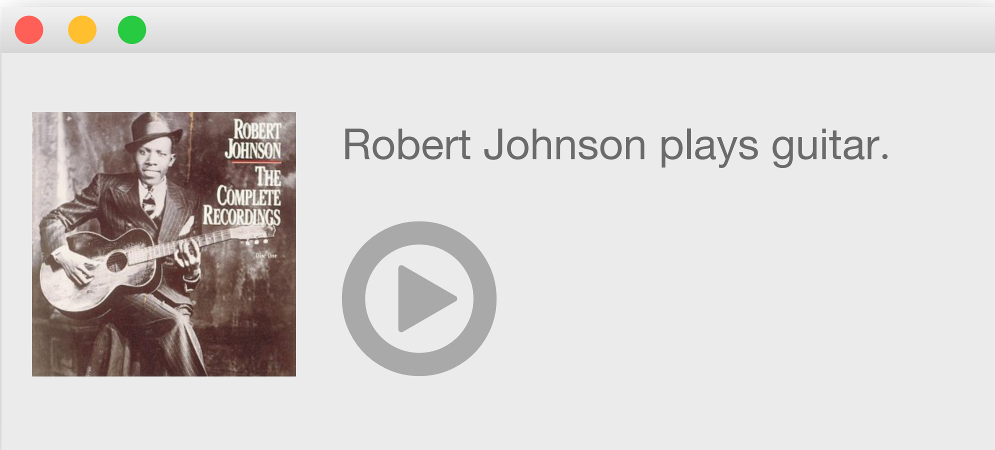
```
what_he_does = ' plays '
his_instrument = 'guitar'
his_name = 'Robert Johnson'
artist_intro = his_name + what_he_does + his_instrument

print(artist_intro)
```

你会发现输出了这样的结果：

```
Robert Johnson plays guitar
```

也许你会觉得无聊，但实际上这段代码加上界面之后是下图这样的，类似于你在音乐播放器里面经常看到的样子。Robert Johnson是著名的美国蓝调吉他手，被称为与魔鬼交换灵魂的人。



注:本图的GUI图形界面采用了 Python 标准库 TKinter 进行实现。

也许你已经注意到了，上面我们说到变量的时候，有些变量被进行不同形式的赋值。我们现在试着在 IDE 中这样做：

```
num = 1
string = '1'

print(num + string)
```

你一定会得到如下的结果，原因是字符串（string）只是Python中的一种数据类型，另一种数据类型则称之为整数（integer），而不同的数据类型是不能够进行合并的，但是通过一些方法可以得到转换。

```
>>> num = 1
>>> string = '1'
>>> print(num + string)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> 
```

插一句，如果你不知道变量是什么类型，可以通过 type() 函数来查看类型。在 IDE 中输入 `print(type(word))`

另外，由于中文注释会导致报错，所以需要在文件开头加一行魔法注释 `#coding:utf-8`，也可以在设置里面找到“File Encodings”设置为 UTF-8。

接下来，我们来转化数据数据类型。我们需要将转化后的字符串储存在另一个变量中，试着输入这些：

```
num = 1
string = '1'
num2 = int(string)

print(num + num2)
```

这样被转换成了同种类型之后，就可以合并这两个变量了。

我们来做一些更有意思的事情，既然字符串可以相加，那么字符串之间能不能相乘？当然可以！输入代码：

```
words = 'words' * 3
print(words)

wordswordswords
```

好，现在我们试着解决一个更复杂的问题：

```
word = 'a loooooong word'
num = 12
string = 'bang!'
total = string * (len(word) - num) #等价于字符串'bang!'*4
print(total)
```

到这里，你就掌握了字符串最基本的用法了，Bang!

字符串的分片与索引

字符串可以通过 string[x] 的方式进行索引、分片，也就是加一个 []。字符串的分片(slice)实际上可以看作是从字符串中找出来你要截取的东西，复制出来一小段你要的长度，储存在另一个地方，而不会对字符串这个源文件改动。分片获得的每个字符串可以看作是原字符串的一个副本。

先来看下面这段代码。如果你对字符串变量后面一些莫名其妙的数字感到困惑和没有头绪的话，不妨对照着代码下面的这个表格来分析。

```
name = 'My name is Mike'

print(name[0])
'M'
print(name[-4])
'M'
print(name[11:14])  # from 11th to 14th, 14th one is excluded
'Mik'
print(name[11:15])  # from 11th to 15th, 15th one is excluded
'Mike'
print(name[5:])
'me is Mike'
print(name[:5])
'My na'
```

name = "	M	y		N	a	m	e		i	s		M	i	k	e	"
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
INDEXING	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

`:`两边分别代表着字符串的分割从哪里开始，并到哪里结束。

以`name[11:14]`为例，截取的编号从第11个字符开始，到位置为14但不包含第14个字符结束。

而像`name[5:]`这样的写法代表着从编号为5的字符到结束的字符串分片。

相反，`name[:5]`则代表着从编号为0的字符开始到编号为5但不包含第5个字符的字符分片。可能容易搞混，可以想象成第一种是从5到最后面，程序员懒得数有多少个所以就省略地写。第二种是从最前面到5，同样是懒得写0，所以就写成了`[:5]`。

好，现在我们试着解决一个更复杂的问题，来做一个文字小游戏叫做——“**找出你朋友中的魔鬼**”。输入代码：

```
word = 'friends'
find_the_evil_in_your_friends = word[0]+word[2:4]+word[-3:-1]
print(find_the_evil_in_your_friends)
```

注：过长的代码段可以使用‘\’来进行换行，属于一行的代码同时会有一个缩进代表是一行的，如果是在IDE中只要按“enter”就能自动换行，很方便吧？

如果运行正常，你就会发现这样的答案：`fiend`
也就发现了朋友中的魔鬼（get到了吗？）。

再来看一个实际项目中的应用，同样是分片的用法。

```
'http://ww1.site.cn/14d2e8ejw1exjogbxdxhj20ci0kuwex.jpg'
'http://ww1.site.cn/85cc87jw1ex23yhwws5j20jg0szmzk.png'
'http://ww2.site.cn/185cc87jw1ex23ynr1naj20jg0t60wv.jpg'
'http://ww3.site.cn/185cc87jw1ex23yyvq29j20jg0t6gp4.gif'
```

在实际项目中切片十分好用。上面几个网址（网址经过处理，所以你是打不开的）是使用 Python 编写爬虫后，从网页中解析出来的部分图片链接，现在总共有500余张附有这样链接的图片要进行下载，也就是说我需要给这500张不同格式的图片（png.jpg,gif）以一个统一的方式进行命名。通过观察规律，决定以链接尾部倒数10个字符的方式进行命名，于是输入代码如下：

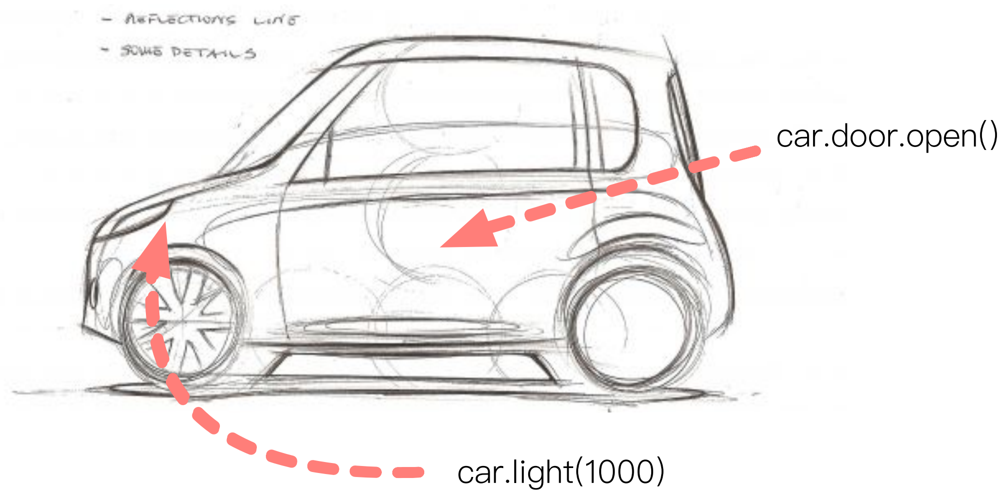
```
url = 'http://ww1.site.cn/14d2e8ejw1exjogbxdxhj20ci0kuwex.jpg'
file_name = url[-10:]

print(file_name)
```

你会得到这样的结果：`0kuwex.jpg`

字符串的方法

Python 是面向对象进行编程的语言，而对象拥有各种功能、特性，专业术语称之为——方法（Method）。为了方便理解，我们假定日常生活中的车是“对象”，即car。然后众所周知，汽车有着很多特性和功能，其中‘开’就是汽车一个重要功能，于是汽车这个对象使用‘开’这个功能，我们在Python 编程中就可以表述成这样：car.drive()



在理解了对象的方法后，我们来看这样一个场景。很多时候你使用手机号在网站注册账户信息，为了保证用户的信息安全性，通常账户信息只会显示后四位，其余的用“*”来代替，我们试着用字符串的方法来完成这一个功能



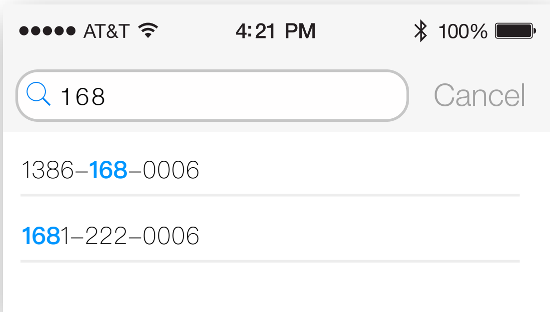
输入代码:

```
phone_number = '1386-666-0006'
hiding_number = phone_number.replace(phone_number[:9],'*' * 9)
print(hiding_number)
```

其中我们使用了一个新的字符串方法 `replace()` 进行“遮挡”。replace方法的括号中，第一个`phone_number[:9]`代表要被替换掉的部分，后面的`'*'*9`表示将要替换成什么字符，也就是把*乘以9，显示9个*。

你会得到这样的结果：`*********0006`

现在我们试着解决一个更复杂的问题，来模拟手机通讯簿中的**电话号码联想功能**



注:在这里只是大致地展示解决思路，真实的实现方法远比我们看到的要复杂

输入代码:

```
search = '168'
num_a = '1386-168-0006'
num_b = '1681-222-0006'

print(search + ' is at ' + str(num_a.find(search)) + ' to '+ str(num_a.find(search) + len(search)) + ' of num_a')
print(search + ' is at ' + str(num_b.find(search)) + ' to '+ str(num_b.find(search) + len(search)) + ' of num_b')
```

你会得到这样的结果，代表了包含168的所有手机号码

```
168 is at 5 to 8 of num_a
168 is at 0 to 3 of num_b
```

字符串格式化符

_____a word she can get what she _____ for.

A.With　　　　B.came

这样的填空题会让我们印象深刻，当字符串中有多个这样的“空”需要填写的时候，我们可以使用.format()进行批处理，它的基本使用方法有如下几种，输入代码：

```
print('{} a word she can get what she {} for.'.format('With','came'))
print('{preposition} a word she can get what she {verb} for'.format(preposition = 'With',verb = 'came'))
print('{0} a word she can get what she {1} for.'.format('With','came'))
```

这种字符串填空的方式使用很广泛，例如下面这段代码可以填充网址中空缺的城市数据：

```
city = input("write down the name of city:")
url = "http://apistore.baidu.com/microservice/weather?citypinyin={}".format(city)
```

注:这是利用百度提供的天气api实现客户端天气插件的开发的代码片段

好了，到这里你就掌握了变量和字符串的基本概念和常用方法。
下一步，我们会继续学习更深一步的循环与函数。

第四章

最基本的魔法
函数

重新认识函数

我们先不谈 Python 中的函数定义，因为将定义放在章节的首要位置，这明显就是懒得把事情讲明白的做法，相信你在阅读其他教材时对这点也深有体会。所以我要说的是，经过第一章的阅读与训练，其实你早已掌握了函数的用法：



print 是一个放入对象就能将结果打印的函数



input 是一个可以让用户输入信息的函数



len 是一个可以测量对象长度的函数



int 是一个可以将字符串类型的数字转换成整数类型的函数

通过观察规律其实不难发现，Python 中所谓的使用函数就是把你要处理的对象放到一个名字后面的括号里就可以了。简单的来说，函数就是这么使用，可以往里面塞东西就得到处理结果。这样的函数在 Python 中还有这些：

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

以最新的3.50版本为例，一共存在68个这样的函数，它们被统称为内建函数（Built-in Functions)。之所以被称之为内建函数，并不是因为还有“外建函数”这个概念，内建的意思是这些函数在3.50版本安装完成后你就可以使用它们，是“自带”的而已。千万不要为这些术语搞晕了头，随着往后学习，我们还能看见更多这样的术语，其实都只是很简单的概念，毕竟在一个专业领域内为了表达准确和高效往往会使用专业术语。

现在你并不必急着把这些函数是怎么用的都搞明白，其中一些内建函数很实用，但是另外一些就不常用，比如涉及字符编码的函数ascii()，bin()，chr()等等，这些都是相对底层的编程设计中才会使用到的函数，在你深入到一定程度的时候才会派的上用场。

附上　Python官网中各个函数介绍的链接：https://docs.Python.org/3/library/functions.html，有兴趣深入了解的话可以看一眼。

开始创建函数

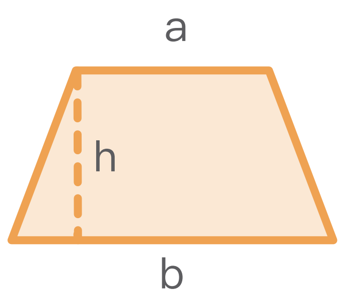
我们需要学会使用已有的函数，更需要学会创建新的函数。自带的函数数量是有限的，想要让 Python 帮助我们做更多的事情，就要自己设计符合使用需求的函数。创建函数也很简单，其实我们在多年前的初中课堂上早已掌握了其原理。

先试着在命令行/终端中进入 Python 环境，输入这样的公式：

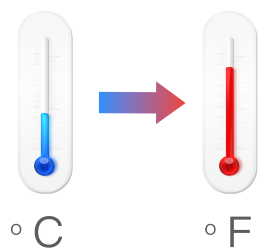
```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 23 2015, 02:52:03)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 1/2*(3+4)*5
17.5
>>> 32*9/5 + 32
89.6
>>>
```

看着有点眼熟吧？第一个是数学的梯形计算公式，而第二个是物理的摄氏度与华氏度的转换公式。

$$S = \frac{(a+b)h}{2}$$



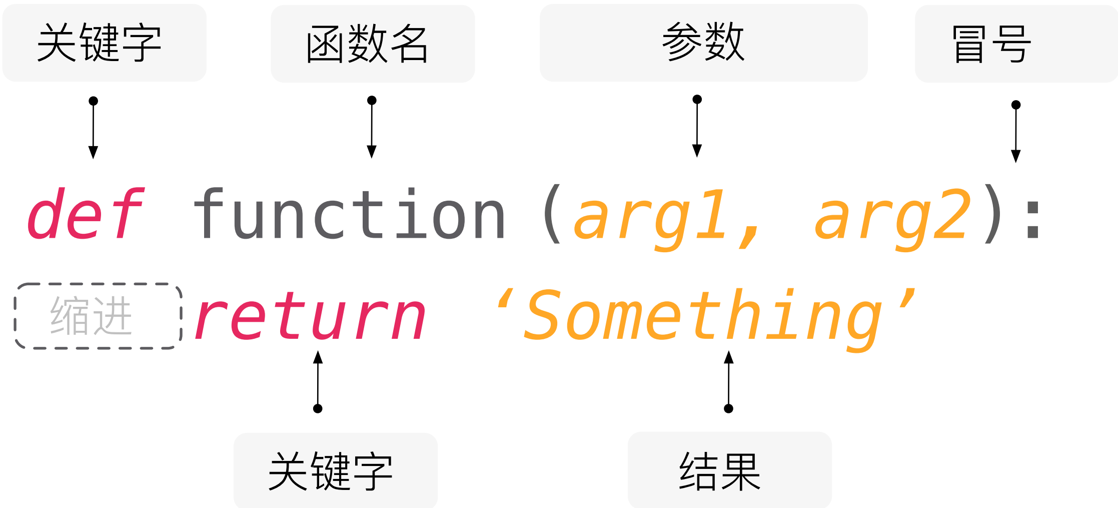
$$F = \frac{9}{5}C + 32$$



函数是编程中最基本的魔法，但同时一切的复杂又都被隐含其中。它的原理和我们学习的数学公式相似，但是并不完全一样，等到后面一点你就知道我为什么这么说了。这里面先介绍几个常见的词：

- def（即 define,定义）的含义是创建函数，也就是定义一个函数。
- arg（即 argument,参数）有时你还能见到这种写法：parameter，二者都是参数的意思但是稍有不同，这里不展开说了。
- return 即返回结果。

好，现在我们读一遍咒语：Define a function named ‘function’ which has two arguments：arg1 and arg2，returns the result——‘Something’ 是不是很易读很顺畅？代码的表达比英文句子更简洁一点：



需要注意的是：

- def 和 return 是关键字（keyword），Python 就是靠识别这些特定的关键字来明白用户的意图，实现更为复杂的编程。像这样的关键字还有一些，在后面的章节中我们会细致讲解；
- 在闭合括号后面的冒号必不可少，而且非常值得注意的是你要使用英文输入法进行输入，否则就是错误的语法，如果你在IDE中输入中文的冒号和括号，会有这样的错误提示：

```
def function（）：
    return 'Something'
```

```
    def function（）：
                  ^
SyntaxError: invalid character in identifier
```

- 如果在IDE中冒号后面回车（换行）你会自动地得到一个缩进。函数缩进后面的语句被称作是语句块（block），缩进是为了表明语句和逻辑的从属关系，是 Python 最显著的特征之一。**很多初学者会忽视缩进问题，导致代码无法成功运行，在这里需要特别注意。**

现在我们看一下之前提到的摄氏度转化公式，按照上面定义函数的方法来实现一遍。我们把摄氏度转化定义为函数fahrenheit_Converter()，那么将输入进去的必然是摄氏度（Celsius）的数值，我们把 C 设为参数，最后返回的是华氏度（fahrenheit）的数值，我们用下面的函数来表达，输入代码：

```
def fahrenheit_converter(C):
    fahrenheit = C * 9/5 + 32
    return str(fahrenheit) + '°F'
```

注:计算的结果类型是int，不能与字符串“°F”相合并，所以需要先用str()函数进行转换

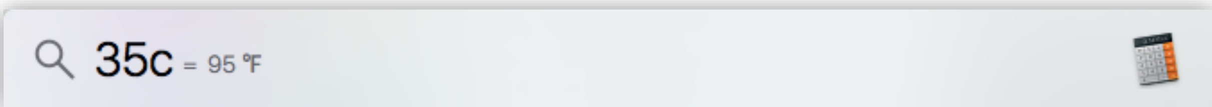
输入完以上代码后，函数定义完成，那么我们开始使用它。我们把使用函数这种行为叫做“调用”（call），你可以简单地理解成你请求 Python 给你帮忙去做一件事情，就像是我们之前学习到的函数 len()一样：“请帮我测量这个（对象）的长度，并将结果打印出来。“

```
lyric_length = len('I Cry Out For Magic!')
print(lyric_length)
```

就像我们使用len()函数一样，下面这段代码意味着——“请使用摄氏度转换器将35摄氏度转换成华氏度，将结果储存在名为C2F的变量并打印出来。”这样我们就完成了函数的调用同时打印了结果。

```
C2F = fahrenheit_converter(35)
print(C2F)
```

对应的结果应该是95.0°F,你可以找一个摄氏度转换器计算一下，下面是我使用Mac 自带的Spotlight 的计算结果



好，到了这里函数的定义和基本用法你就已经了解，在很长一段时间内你知道上面所讲的这些内容就基本够用了，但为了让你在深入使用函数的时候不产生困惑和挣扎，接下来我们试着解决一个更复杂的问题。

我们把刚才的函数按照如下进行修改：

```
def fahrenheit_converter(C):
    fahrenheit = C * 9/5 + 32
    print(str(fahrenheit) + '°F')
```

怎么样？看上去很相似吧？没错，我们仅仅就是把最后一行的 return 换成了 print 函数，一个很小的改动，而且似乎 IDE 也并没有对语法进行报错预警，那么我们来试一下调用函数会是什么情况吧：

```
C2F = fahrenheit_converter(35)
print(C2F)
```

运行起来的结果是这样的：

```
95.0°F
None
```

为什么会这样？

其实，得到这样的结果是因为 print 是一个函数，并非关键字（如果你的print不是函数那说明你的版本还停留在2.x系列，现在就赶紧安装3.0以上的版本！）。如果你足够细心的话可以发现，在我的IDE中，虽说 print 与 return 它们都是蓝色，但实际是有区分的：一个是正常体，一个是斜体。return作为关键字在函数中起到了返回值的作用，而 print 顾名思义，只是在函数中展示给我们打印的结果，**是为人类设计的函数**。因此上面的 95.0°F 实际上是**调用函数后产生的数值**，而下面的 None 正是此时变量 C2F 中所被返回到的数值——什么都没有，就因为没有关键字 return 。这就好比你对着一个人喊了一声他的名字（call），他只是“哎”地回应你一声，这是因为你并没有告诉他该做什么（return）。

没有 return 也没关系，不代表没有用，在 Python 中 return 是可选的（optional），这意味着你可以不用写 return 也可以顺利地定义一个函数并使用，只不过返回值是‘None’罢了。在后面我们还能见到不同使用方式的函数，这里只需要记住函数的基本设定即可。

在前面我们提到过，定义一个函数使用 def（define），同时我们还能在各种教材不同版本的翻译中看到声明（declare）这个词，我们不难推测，从表达的目的上来说他们是一样的，而对于有其他语言基础的人来说这两个词意味着两种不同的行为。其实没关系，在 Python 中 definition 和 declaration 是一体的，在这里说明仅仅是为了解答有此困惑的人，深究则无意。

练习题

1、初级难度：设计一个重量转换器，输入以“g”为单位的数字后返回换算成“kg”的结果
2、中级难度：设计一个求直角三角形斜边长的函数（两条直角边为参数，求最长边）
如果直角边边长分分别为3和4，那么返回的结果应该像这样：

```
The right triangle third side's length is 5.0
```

建议你动手练习一次，然后在微信公众号中回复“**函数**”获得答案，微信公众号是：easypython

扫码查看练习题答案

传递参数与参数类型

前面大刀阔斧地说了关于函数定义和使用，在这一节我们谈论一些细节但是重要的问题——参数。对于在一开始就设定了必要参数的函数来说，我们打出函数的名称并向括号中传递参数实现对函数的调用（call），只要把参数放进函数的括号中即可，就像是这样：

```
fahrenheit_converter(35)
fahrenheit_converter(15)
fahrenheit_converter(0)
fahrenheit_converter(-3)
```

事实上，传递参数的方式有两种：

位置参数　（positional argument）

关键词参数（keyword argument）

现在从似乎被我们遗忘的梯形的数学公式开始入手，首先还是创建函数。

我们把函数的名称定为trapezoid_area，也就是梯形面积，设定参数为base_up（上底），base_down（下底），height（高）每一个都用英文输入法的逗号隔开。梯形的面积需要知道这三个值才能求得，因此对于构造梯形面积函数来说，这三个参数缺一不可。

```
def trapezoid_area(base_up, base_down, height):
    return 1/2 * (base_up + base_down) * height
```

接下来我们开始调用函数。

```
trapezoid_area(1,2,3)
```

不难看出，填入的参数1，2，3分别对应着参数base_up，base_down和height。这种传入参数的方式被称作为**位置参数**。

接着是第二种传入方式：

```
trapezoid_area(base_up=1, base_down=2, height=3)
```

更直观地，在调用函数的时候，我们将每个参数名称后面赋予一个我们想要传入的值。这种以名称作为一一对应的参数传入方式被称作是**关键词参数**。

想一想去餐厅预约与就餐的流程，找到你预约的座位一般是用你留下的姓名，你就是一个参数，你会被按照姓名的方式传入你预定的座位，这个就是关键词参数传入；接下来是上菜，菜品按照你的座位号的方式来传入你的桌子，而这就相当于是位置传入参数。

也许你现在想不太明白这种传入的方式有何作用，没有关系，在后面我们会和其他知识再一并进行讲解的，到那时你就会对参数的传入方式有更高层次的认识。

避免混乱的最好方法就是先制造混乱，我们试着解决一个更复杂的问题，按照下面几种方式调用函数并打印结果：

```
trapezoid_area(height=3, base_down=2, base_up=1)   # RIGHT!

trapezoid_area(height=3, base_down=2, 1)            # WRONG!

trapezoid_area(base_up=1, base_down=2, 3)           # RIGHT!

trapezoid_area(1, 2, height=3)                      # RIGHT!
```

- 第一行的函数参数按照反序传入，因为是关键词参数，所以并不影响函数正常运作；
- 第二行的函数参数反序传入，但是到了第三个却变成了位置参数，遗憾的是这种方式是错误的语法，因为如果按照位置来传入，最后一个应该是参数height的位置。但是前面height已经按照名称传入了值3，所以是冲突的。
- 第三行的函数参数正序传入，前两个是以关键词的方式传入，最后一个以位置参数传入，这个函数是可以正常运行的；
- 第四行的函数参数正序传入，前两个是以位置的方式传入，最后一个以关键词参数传入，这个函数是可以正常运行的。

> 注:正确运行的结果应该是4.5，也就是这个梯形的面积。

我们现在给一组变量赋值，然后再调用函数：

```
base_up = 1
base_down = 2
height = 3

trapezoid_area(height, base_down, base_up)
```

然而这次函数调用的结果应该是2.5，为什么？

如果你有这样的困惑，说明你已经被参数的命名和变量的命名搞晕，我们来把这两者区分清晰。首先，我们在定义函数的时候会定义参数的名称，其主要作用就是方便我们在使用函数时指导我们将传入什么参数，它们从哪里来，是什么类型等，提供与使用函数使用相关的上下文。下面这段代码也许能够帮助你理解函数来自参数名称的困扰：

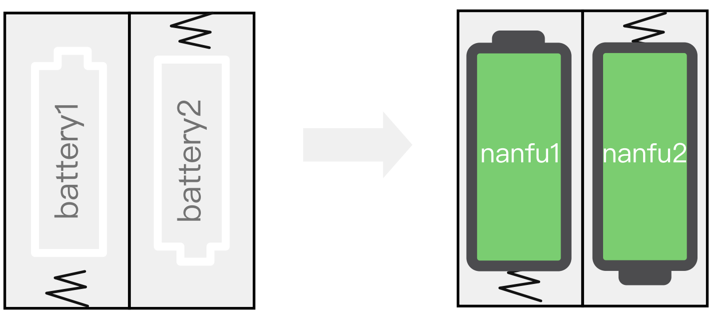
```
def flashlight (battery1, battery2):
    return 'Light!'
```

我们定义一个叫做手电筒（flashlight）的函数，它需要两个参数 battery1 和 battery2 意为电池。这时候你去商店买电池，买回了两节600毫安时的南孚电池，于是：

```
nanfu1 = 600
nanfu2 = 600

flashlight(nanfu1, nanfu2)
```

看明白了吗？南孚是电池的一种，是可以让手电筒发光的东西，将南孚电池放入就意味着我们放入了手电筒所需的电池，换句话说，nanfu1，nanfu2是变量，同时也是满足能够传入的函数的flashlight函数的参数，传入后就代替了原有的battery1和battery2且传入方式仍旧是位置参数传入。battery1和battery2只是形式上的占位符，表达的意思是函数所需的参数应该是和电池即battery有关的变量或者是对象。



最后关于参数，我们来学习一个小秘密。本章一开始就说过，一开始设定好的参数在调用时缺一不可，我们来验证一下：

输入代码：

```
trapezoid_area(1,2)
```

你会看到这样的报错：

```
TypeError:trapezoid_area()
missing 1 required positional
argument: ‘height’
```

嗯，似乎是这样，为什么要用似乎？先来试试下面的代码吧！输入代码：

```
print('   *','  * *',' * * *','   |  ')
```

并没有发现什么异常！

试试这样！

```
print('   *','  * *',' * * *','   |  ',sep = '\n')
```

好神奇！我得到了一棵圣诞树！

```
   *
  * *
 * * *
   |
```

你看到的这个魔法就是我们将要提到的神奇的默认参数。默认参数是可选的，这意味着即使你上来不给它传入什么东西函数还是可以正常运作。

你只需要这样输入代码：

```
def trapezoid_area(base_up, base_down, height=3):
    return 1/2 * (base_up + base_down) * height
```

给一个参数设定默认值非常简单，我们只需要在**定义参数的时候给参数赋值即可**。这个也许跟传入参数的方式有点像，但是千万别记混了！这可是在定义的时候做的事情！这样一来，我们只需要传入两个参数就可以正常进行了：

```
trapezoid_area(1, 2)
```

你肯定会疑惑，如果设定默认值的话，那么所有梯形的高岂不是都固定成3了啊？然而并没有，默认值的理念就是让使用函数尽可能的简单、省力。正如同我们安装软件都会有默认目录，但是如果你又想安装在其他地方，你可以选择自定义修改。之前看到的print函数的小把戏也正是如此，print的可选参数sep（意为每个打印的结果以...分开）的默认值为‘ ’空格，但是我们将其重新传入‘/n’也就是换行的意思，一句话说，也就是将每个打印的数以换行符号进行分割。下面我们来调用自己的参数：

```
trapezoid_area(1, 2, height=15)
```

只需要传入我们想要的值就可以了，就是这么简单。

默认值并非是你掌握参数使用的必要知识，却是能帮助我们节省时间的小技巧。在实际项目中也经常会看见这样：

```
requests.get(url, headers=header)
```

> 注:这个是在请求网站时 header，可填可不填

```
img.save(img_new, img_format, quality=100)
```

> 注:这是在给图片加水印的时候默认的水印质量是100

设计自己的函数

到了这里，我们应该可以十分有自信地设计一个符合自己项目需求的函数了，我们将上面各种所有知识进行整合，来设计一个简易的敏感词过滤器，不过在这之前先来认识一个新的函数——open。

这个函数使用起来很简单，只需要传入两个参数就可以正常运转了：文件的完整路径和名称，打开的方式。

先在桌面上创建一个名为 text.txt 的文件。WIndows 用户在桌面点击右键唤出菜单创建即可，Mac 用户则打开 Pages 创建文件后点击导出格式选择 txt 格式即可。现在我们使用 open 函数打开它：

```
open('/Users/Hou/Desktop/text.txt')
```

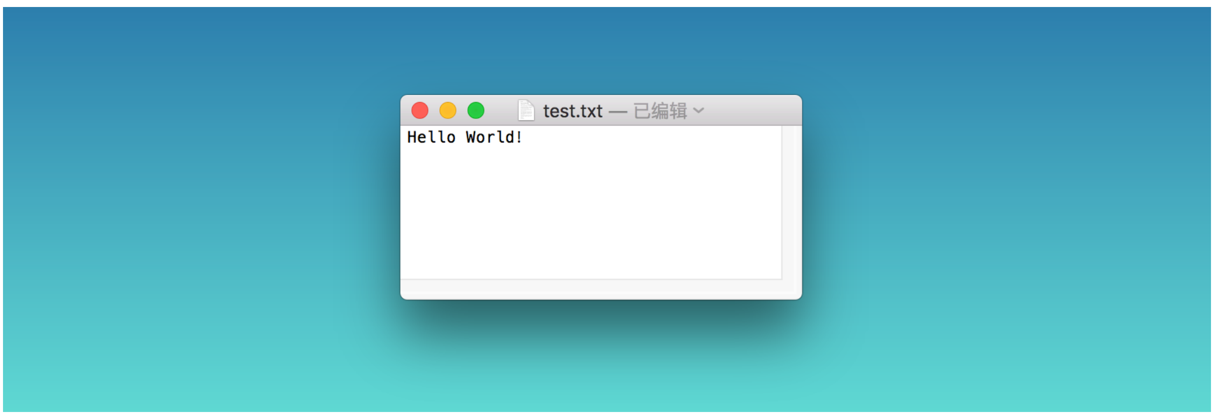
如果是 Windows 用户，应该像这样写你的路径：

```
open('C://Users/Hou/Desktop/')
```

如果你照着代码敲入的话其实这时候文件应该已经是打开的了，但是...貌似我们看不出来，所以，我们再认识一个新的方法——write。在第一章我们已经提到过如何使用方法（如果你现在困惑函数和方法到底是什么关系的话，为了顺利地往后进行，我可以告诉你方法就是函数的一种，只不过在不同的位置而已，使用原理和函数非常相似），在这里我们就照抄第三章的 replace 用法来学着使用 write 方法：

```
file = open('/Users/Hou/Desktop/text.txt','w')
file.write('Hello World')
```

写完后我们运行程序看看效果：



掌握了 open 与 write 的基本用法之后，我们就可以开始着手设计函数了，需求是这样的：传入参数 name 与 msg 就可以控制在桌面写入的文件名称和内容的函数 text_create，并且如果当桌面上没有这个可以写入的文件时，那么就要创建一个之后再写入。现在我们开搞吧！

```
1. def text_create(name, msg):
2.     desktop_path = '/Users/Hou/Desktop/'
3.     full_path = desktop_path + name + '.txt'
4.     file = open(full_path,'w')
5.     file.write(msg)
6.     file.close()
7.     print('Done')
8. text_create('hello','hello world') # 调用函数
```

我们来逐行解释这段代码。

第一行：定义函数的名称和参数；

第二行：我们在最开始知道，open 函数要打开一个完整的路径，所以首先是桌面路径；

第三行：我们给文件起什么名字，就是要传入的参数加上桌面路径再加上后缀就是完整的文件路径了；

第四行：打开文件，`'w'`参数代表作为写入模式，意思是：如果没有就在该路径创建一个有该名称文本，有则追加覆盖文本内容；

第五行：写入传入的参数 msg，即内容；

第六行：关闭文本。

这样一来敏感词过滤器的第一部分我们就完成了。顺带一提，这个函数就是我们在前面提及到的并不需要 return 也能发挥作用的函数，最后的 print 仅仅是为了表明上面的所有语句均已执行，一个提示而已。接下来我们实现第二部分，敏感词过滤，需求是这样的：定义一个为函数text_filter的函数，传入参数 word，cencored_word 和changed_word实现过滤，敏感词cencored_word默认为‘lame’，替换词 changed_word默认为‘Awesome’。现在继续：

```
def text_filter(word,censored_word = 'lame',changed_word = 'Awesome'):
    return word.replace(censored_word, changed_word)
text_filter('Python is lame!')      # 调用函数
```

这个函数就简单的多了，第一行我们按照设定默认参数的方式来定义函数，第二行直接返回使用replace处理后的结果。现在两个函数均已完成，本着低风险的原则，你可以尝试调用一下函数看看返回结果。

现在我们试着解决一个更复杂的问题，把两个函数进行合并：创建一个名为 text_censored_create 的函数，功能是在桌面上创建一个文本可以在其中输入文字，但是如果信息中含有敏感词的话将会被默认过滤后写入文件。其中文本的文件名参数为 name，信息参数为 msg，你可以先试着自己写一下，写完了再对照看下：

```
def censored_text_create(name, msg):
    clean_msg = text_filter(msg)
    text_create(name,clean_msg)
censored_text_create('Try','lame!lame!lame!') # 调用函数
```

我们使用第一个函数将传入的 msg 进行过滤后储存在名为 clean_msg 的变量中，再将传入的 name 文件名参数和过滤好的文本 clean_msg 作为参数传入函数 text_create 中，结果我们会得到过滤后的文本。

完成之后，你就会得到一个文本过滤器了！

在本章中我只是借助数学阐明了函数的运作方式而已，但是如果你确实需要解决许多数学上的问题，在这里我可以给你一个基本的参考表格，至于怎么用，多尝试就知道了。一定要敢于尝试，毕竟电脑也不会因为你的一行代码而爆炸。

假设 a=10, b=20，则运算示例如下：

	描述	实例
+	加 – 两个对象相加	a + b 输出结果 30
–	减 – 得到负数或是一个数减去另一个数	a - b 输出结果 -10
*	乘 – 两个数相乘或是返回一个被重复若干次的字符串	a * b 输出结果 200
/	除 – x除以y	b / a 输出结果 2
%	取模 – 返回除法的余数	b % a 输出结果 0
**	幂 – 返回x的y次幂	a**b 为10的20次方， 输出结果 100000000000000000000
//	取整除 – 返回商的整数部分	9//2 输出结果 4 , 9.0//2.0 输出结果 4.0

第五章

循环与判断

逻辑控制与循环

逻辑判断——True & False

逻辑判断是编程语言最有意思的地方，如果要实现一个复杂的功能或程序，逻辑判断必不可少。if-else 结构就是常见的逻辑控制的手段，当你写出这样的语句的时候，就意味着你告诉了计算机什么时候该怎么做，或者什么是不用做的。学完了前面几章内容之后，现在的你也许早已对逻辑控制摩拳擦掌、跃跃欲试，但是在这之前我们需要先了解逻辑判断的最基本准则——布尔类型（Boolean Type）。

在开始前，想强调一点，如果你怀疑自己的逻辑能力，进而对本章的内容感到畏惧，请不要担心，我可以负责任地说，没有人是"没有逻辑的"，正如我们可以在极其复杂的现实世界中采取各种行动一样，你所需要的只不过是一些判断的知识和技巧而已。

布尔类型（Boolean）的数据只有两种，True 和 False（需要注意的是首字母大写）。人类以真伪来判断事实，而在计算机世界中真伪对应着的则是1和0。

接下来我们打开命令行/终端进入 Python 环境，敲入这些代码，或者直接在 PyCharm 中选择 Python Console ，这样会更方便展示结果。True & False 这一小节的内容我们都在命令行/终端环境里输入代码。



注：此处使用命令行/终端只为更快展现结果，在IDE返回布尔值仍旧需要使用 print 函数来实现。

输入代码：

```
1>2
1<2<3
42 != '42'
'Name' == 'name'
'M' in 'Magic'
number = 12
number is 12
```

我们每输入一行代码就会立即得到结果，这几行代码的表达方式不同，但是返回结果却只有 True 和 False 这两种布尔类型，因此我们称**但凡能够产生一个布尔值的表达式为布尔表达式（Boolean Expressions）。**

```
1 > 2                  # False
1 < 2 <3               # True
42 != '42'             # True
'Name' == 'name'       # False
'M' in 'Magic'         # True
number = 12
number is 12           # True
```

可以看到，上面这些能够产生布尔值的方法或者公式不尽相同，那么我们来一一讲解这些运算符号的意义和用法。

比较运算（Comparison）

对于比较运算符，顾名思义，如果比较式成立那么则返回 True，不成立则返回 False。

比较运算符（Comparison Operators）	
==	左右两边等值的时候返回 True
!=	左右两边不相等时返回 True
>	左边大于右边的时候返回True
<	左边小于右边的时候返回True
<=	左边小于或等于右边的时候返回True
>=	左边大于或等于右边的时候返回True

除了一些在数学上显而易见的事实之外，比较运算还支持更为复杂的表达方式，例如：

多条件的比较。先给变量赋值，并在多条件下比较大小：

```
middle = 5
1 < middle < 10
```

变量的比较。将两个运算结果储存在不同的变量中，再进行比较：

```
two = 1 + 1
three = 1 + 3
two < three
```

字符串的比较。其实就是对比左右两边的字符串是否完全一致，下面的代码就是不一致的，因为在 Python 中有着严格的大小写区分：

```
'Eddie Van Helen' == 'eddie van helen'
```

两个函数产生的结果进行比较：比较运算符两边会先行调用函数后再进行比较，其结果等价于 `10 > 19` ：

```
abs(-10) > len('length of this word')
```

> 注：abs( )是一个会返回输入参数的绝对值的函数。

比较运算的一些小问题

不同类型的对象不能使用“<,>,<=,>=”进行比较，却可以使用’==’和’!=’，例如字符串和数字：

```
42 > 'the answer'     无法比较
42 == 'the answer'     #False
42 != 'the answer'     #True
```

需要注意的是，浮点和整数虽是不同类型，但是不影响到比较运算：

```
5.0 == 5                #True
3.0 > 1                #True
```

你可能会有一个疑问，“为什么 1 = 1 要写作 1 == 1 ?”，前面提及过 Python 中的符号在很多地方都和数学中十分相似，但又不完全一样。“=”在 Python 中代表着赋值，并非我们熟知的“等于”。所以，“1 = 1” 这种写法并不成立，并且它也不会给你返回一个布尔值。使用“= =”这种表达方式，姑且可以理解成是表达两个对象的值是相等的，这是一种约定俗成的语法，记得就可以了。

比较了字符串、浮点、整数...还差一个类型没有进行比较：布尔类型，那么现在实验一下：

```
True > False
True + False > False + False
```

这样的结果又怎么理解呢？还记得前面说过的吗，True 和 False 对于计算机就像是1和0一样，如果在命令行中敲入True + True + False 查看结果不难发现，True = 1， False = 0 也就是说，上面这段代码实际上等价于：

```
1 > 0
1 + 0 > 0 + 0
```

至于为什么是这样的原因，我们不去深究，还是记得即可。

最后一个小的问题，如果在别的教材中看到类似 1<>3 这种表达式也不要大惊小怪，它其实与 1!=3 是等价的，仅仅知道就可以，并不是要让你知道“茴字的四种写法”。

成员运算符与身份运算符（Membership & Identify Operators）

成员运算符和身份运算符的关键词是 in 与 is。把 in 放在两个对象中间的含义是，测试前者是否存在于 in 后面的集合中。说到集合，我们先在这里介绍一个简单易懂的集合类型——列表（List）。

字符串、浮点、整数、布尔类型、变量甚至是另一个列表都可以储存在列表中，列表是非常实用的数据结构，在后面会花更多篇幅来讲解列表的用法，这里先简单了解一下。

创建一个列表，就像是创建变量一样，要给它起个名字：

```
album = []
```

此时的列表是空的，我们随便放点东西进去，这样就创建了一个非空的列表：

```
album = ['Black Star','David Bowie',25,True]
```

这个列表中所有的元素是我们一开始放好的，那当列表创建完成后，想再次往里面添加内容怎么办？使用列表的 append 方法可以向列表中添加新的元素，并且使用这种方式添加的元素会自动地排列到列表的尾部：

```
album.append('new song')
```

接着就是列表的索引，如果在前面的章节你很好地掌握了字符串的索引，相信理解新的知识应该不难。下面代码的功能是打印列表中第一个和最后一个元素：

```
print(album[0],album[-1])
```

接下来我们使用 in 来测试字符串'Black Star' 是否在列表 album 中。如果存在则会显示 True，不存在就会显示 False 了：

```
'Black Star' in album
```

是不是很简单？正如前面看到的那样，in 后面是一个集合形态的对象，字符串满足这种集合的特性，所以可以使用 in 来进行测试。

接下来再来讲解 is 和 is not，它们是表示身份鉴别（Identify Operator）的布尔运算符，in 和 not in 则是表示归属关系的布尔运算符号（Membership Operator）。

在 Python 中任何一个对象都要满足身份（Identity）、类型（Type）、值（Value）这三个点，缺一不可。is 操作符号就是来进行身份的对比的。试试输入这段代码：

```
the_Eddie = 'Eddie'
name = 'Eddie'
the_Eddie == name
the_Eddie is name
```

你会发现在两个变量一致时，经过 is 对比后会返回 True。

其实在 Python 中任何对象都可判断其布尔值，除了 0、None 和所有空的序列与集合（列表，字典，集合）布尔值为 False 之外，其它的都为 True ，我们可以使用函数 bool() 进行判别：

```
bool(0)       #False
bool([])      #False
bool('')      #False
bool(False)   #False
bool(None)    #False
```

可能有人不明白，为什么一个对象会等于 None。还记得在函数那章的敏感词过滤器的例子吗？在定义函数的时候没有写 return 依然可以使用，但如果调用函数，企图把根本就不存在的“返回值”储存在一个变量中时，变量实际的赋值结果将是 None。

当你想设定一个变量，但又没想好它应该等于什么值时，你就可以这样：

```
a_thing = None
```

布尔运算符（Boolean Operators）

and、or 用于布尔值的之间的运算，具体规则如下：

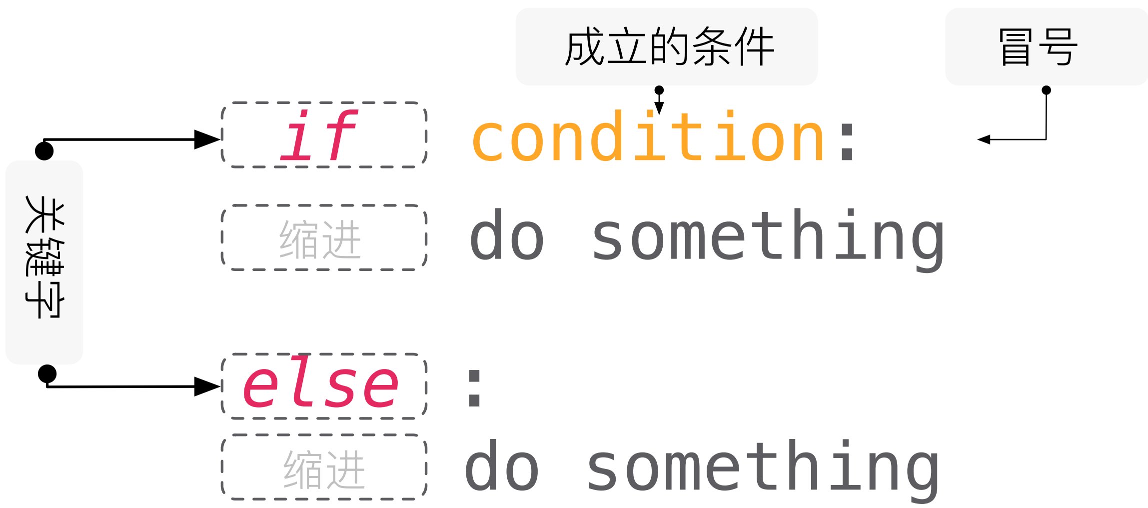
运算符	描述
not x	如果 x 是 True，则返回 False，否则返回 True。
x and y	and 表示“并且”，如果 x 和 y 都是 True，则返回 True；如果 x 和 y 有一个是 False，则返回 False。
x or y	or 表示“或者”，如果 x 或 y 有其中一个是 True，则返回 True；如果 x 和 y 都是 False，则返回 False。

and 和 or 经常用于处理复合条件，类似于 1 < n < 3 ，也就是两个条件同时满足。

```
1 < 3 and 2 < 5   #True
1 < 3 and 2 > 5   #False
1 < 3 or 2 > 5    #True
1 > 3 or 2 > 5    #False
```

条件控制

条件控制其实就是 if…else 的使用。先来看下条件控制的基本结构：



用一句话概括 if…else 结构的作用：**如果…条件是成立的，就做….；反之，就做…**

所谓条件（condition）指的是成立的条件，即是**返回值为 True 的布尔表达式**。知道了这点后使用起来应该不难。

我们结合函数的概念来创建这样一个函数，逐行分析它的原理：

```
1. def account_login():
2.     password = input('Password:')
3.     if password == '12345':
4.         print('Login success!')
5.     else:
6.         print('Wrong password or invalid input!')
7.         account_login()
8. account_login()
```

第1行：定义函数，并不需要参数；

第2行：使用 input 获得用户输入的字符串并储存在变量 password 中；

第3、4行：设置条件，如果用户输入的字符串和预设的密码12345相等时，就执行打印文本‘Login success!’；

第5、6行：反之，一切不等于预设密码的输入结果，全部会执行打印错误提示，并且再次调用函数，让用户再次输入密码；

第7行：运行函数。

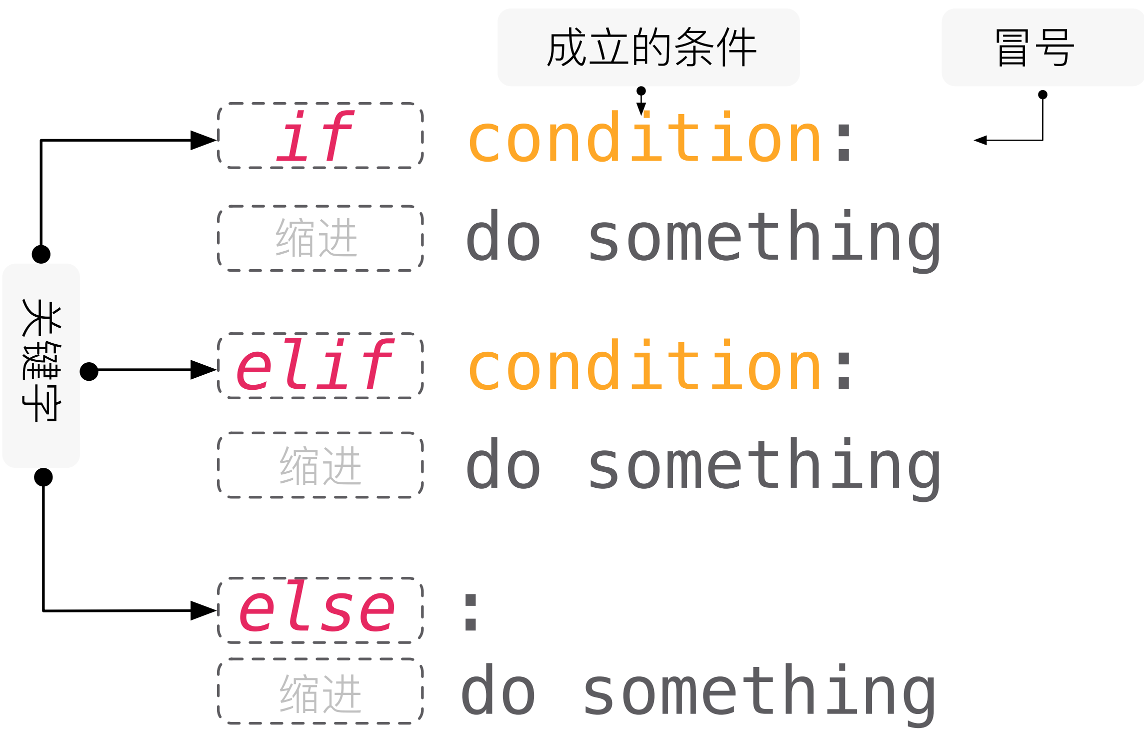
第8行：调用函数

值得一提的是，如果 if 后面的布尔表达式过长或者难于理解，可以采取给变量赋值的办法来储存布尔表达式返回的布尔值 True 或 False。因此上面的代码可以写成这样：

```
def account_login():
    password = input('Password:')
    password_correct = password == '12345'     #HERE!
    if password_correct:
        print('Login success!')
    else:
        print('Wrong password or invalid input!')
        account_login()
account_login()
```

一般情况下，设计程序的时候需要考虑到逻辑的完备性，并对用户可能会产生困扰的情况进行预防性设计，这时候就会有多条件判断。

多条件判断同样很简单，只需在 if 和else 之间增加上 elif，用法和 if 是一致的。而且条件的判断也是依次进行的，首先看条件是否成立，如果成立那么就运行下面的代码，如果不成立就接着顺次地看下面的条件是否成立，如果都不成立则运行 else 对应的语句。



接下来我们使用 elif 语句给刚才设计的函数增加一个重置密码的功能：

```
1. password_list = ['*#*#','12345']
2. def account_login():
3.     password = input('Password:')
4.     password_correct = password == password_list[-1]
5.     password_reset = password == password_list[0]
6.     if password_correct:
7.         print('Login success!')
8.     elif password_reset:
9.         new_password = input('Enter a new password:')
10.        password_list.append(new_password)
11.        print('Your password has changed successfully!')
12.        account_login()
13.    else:
14.        print('Wrong password or invalid input!')
15.        account_login()
16.account_login()
```

第1行：创建一个列表，用于储存用户的密码、初始密码和其他数据（对实际数据库的简化模拟）；

第2行：定义函数；

第3行：使用 input 获得用户输入的字符串并储存在变量 password 中；

第4行：当用户输入的密码等于密码列表中最后一个元素的时候（即用户最新设定的密码），登录成功；

第5~9行：当用户输入的密码等于密码列表中第一个元素的时候（即重置密码的“口令”）触发密码变更，并将变更后的密码储存至列表的最后一个，成为最新的用户密码；

第10行：反之，一切不等于预设密码的输入结果，全部会执行打印错误提示，并且再次调用函数，让用户再次输入密码；

第11行：调用函数。

在上面的代码中其实可以清晰地看见代码块（Code Block）。代码块的产生是由于缩进，也就是说，具有相同缩进量的代码实际上是在共同完成相同层面的事情，这有点像是编辑文档时不同层级的任务列表。

循环（Loop）

for 循环

我们先来看一个例子，输入代码：

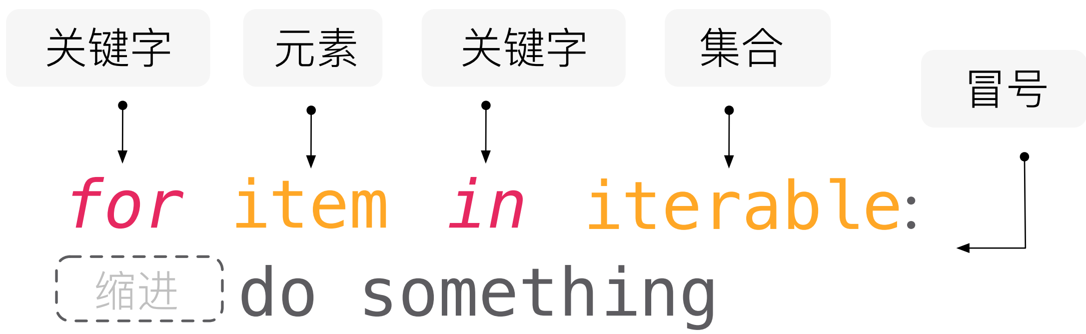
```
for every_letter in 'Hello world':
    print(every_letter)
```

```
H
e
l
l
o

w
o
r
l
d
```

这两行代码展示的是：使用 for 循环打印出“hello world”这段字符串中的每一个字符。for 循环作为编程语言中最强力的特性之一，能够帮助我们做很多重复性的事情，比如批量命名、批量操作等等。

把 for 循环所做的事情概括成一句话就是：**于…其中的每一个元素，做…事情**。



- for 是关键词，而后面紧接着的是一个可以容纳“每一个元素”的变量名称，至于变量起什么名字自己定，但切记不要和关键词重名。
- 在关键词 in 后面所对应的一定是具有“可迭代的”（iterable）或者说是像列表那样的集合形态的对象，即可以连续地提供其中的每一个元素的对象。

为了更深入了解 for 循环，试着思考下面这个问题，如何打印出这样的结果?

```
1 + 1 = 2
2 + 1 = 3
.
.
10 + 1 = 11
```

这需要用到一个内置函数——range。我们只需要在 range 函数后面的括号中填上数字，就可以得到一个具有连续整数的序列，输入代码:

```
for num in range(1,11): #不包含11，因此实际范围是1～10
    print(str(num) + ' + 1 =',num + 1)
```

这段代码表达的是：将1～10范围内的每一个数字依次装入变量 num 中，每次展示一个 num +1 的结果。在这个过程中，变量 num 被循环赋值10次，你可以理解成等同于:

```
num = 1
print(str(num) + ' + 1 =',num + 1)
num = 2
print(str(num) + ' + 1 =',num + 1)
                :
                :
num = 10
print(str(num) + ' + 1 =',num + 1)
```

现在我们试着解决更复杂的问题，把 for 和 if 结合起来使用。实现这样一个程序：歌曲列表中有三首歌“Holy Diver, Thunderstruck, Rebel Rebel”，当播放到每首时，分别显示对应的歌手名字“Dio, AC/CD, David Bowie”。

代码如下:

```
songslist = ['Holy Diver', 'Thunderstruck', 'Rebel Rebel']
for song in songslist:
    if song == 'Holy Diver':
        print(song,' - Dio')
    elif song == 'Thunderstruck':
        print(song,' - AC/DC')
    elif song == 'Rebel Rebel':
        print(song,' - David Bowie')
```

在上述代码中，将 songslist 列表中的每一个元素依次取出来，并分别与三个条件做比较，如果成立则输出相应的内容。

嵌套循环

在编程中还有一种常见的循环，被称之为嵌套循环（Nested Loop），其实这种循环并不复杂而且还非常实用。我们都学过乘法口诀表，又称“九九表”，接下来我们就用嵌套循环实现它：

乘法口诀表

X	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	47	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

```python
for i in range(1,10):
    for j in range(1,10):
        print('{} X {} = {}'.format(i,j,i*j))
```

正如代码所示，这就是嵌套循环。通过观察，我们不难发现这个嵌套循环的原理：最外层的循环依次将数值 1～9 存储到变量 i 中，变量 i 每取一次值，内层循环就要依次将 1～9 中存储在变量 j 中，最后展示当前的 i、j 与 i*j 的结果。如果忘了 {} 的用法，可以往回翻第三章最后一页看看。

while 循环

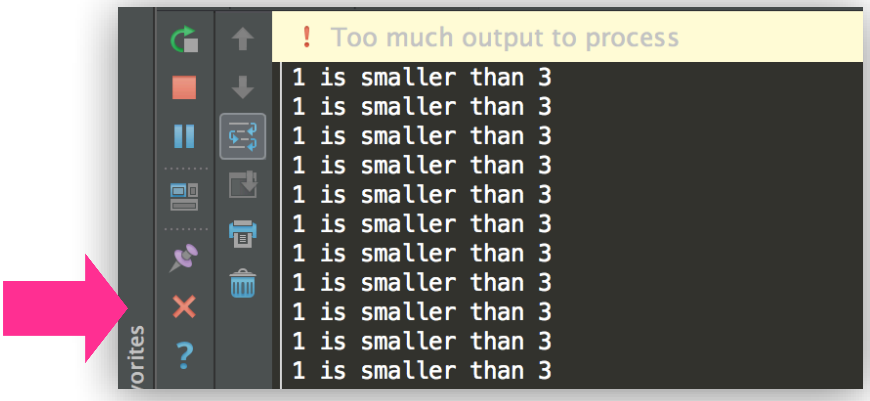
Python 中有两种循环，第一种 for 循环我们已经介绍过了，第二种则是 while 循环。它们的相同点在于都能循环做一件重复的事情，不同点在于 for 循环会在可迭代的序列被穷尽的时候停止，while则是在条件不成立的时候停止，因此 while 的作用概括成一句话就是：**只要…条件成立，就一直做…。**



看一个简单的例子：

```
while 1 < 3:
    print('1 is smaller than 3')
```

在这里先行提醒一下，一定要记得及时停止运行代码！



注：在终端或者命令行中按 Ctrl + C 停止运行，在 PyCharm 中则点击红色的X停止

因为在 while 后面的表达式是永远成立的，所以 print 会一直进行下去直至你的 cpu 过热。这种条件永远为 True 的循环，我们称之为死循环（Infinite Loop）。

但如果 while 循环不能像 for 循环那样，在集合被穷尽之后停下来，我们又怎么样才能控制 while 循环呢？其中一种方式就是：**在循环过程中制造某种可以使循环停下来的条件**，例如：

```
count = 0
while True:
    print('Repeat this line !')
    count = count + 1
    if count == 5:
        break
```

在上面这段代码中，有两个重要的地方，首先是我们给一个叫 count 的变量赋值为 0，其目的是计数。我们希望在循环次数为 5 的时候停下来。接下来的是 break ，同样作为关键词写在 if 下面的作用就是告诉程序在上面条件成立的时候停下来，仅此而已。

然而你也一定发现了什么奇怪的地方，没错，就是这个 count = count + 1！其实我已经不止一次强调过编程代码和数学公式在某些地方很相似，但又不完全相同，而这又是一个绝好的例子。首先在 Python 中“=”并非是我们熟知的“等于”的含义，所以我们不必按照数学公式一样把重复的变量划掉。其次 count 被赋值为 0，count = count +1 意味着 count 被重新赋值！等价于 count = 0 + 1，随着每次循环往复，count 都会在上一次的基础上重新赋值，都会增加+1，直至 count 等于5 的时候 break，跳出最近的一层循环，从而停下来。

利用循环增加变量其实还是一个挺常见的技巧，随着循环不仅可以增加，还可以随着循环减少（n = n – 1），甚至是成倍数增加(n = n*3)。

除此之外，让 while 循环停下来的另外一种方法是：**改变使循环成立的条件。**为了解释这个例子，我们在前面登录函数的基础上来实现，给登录函数增加一个新功能：输入密码错误超过3次就禁止再次输入密码。你可以尝试写一下，答案在下一页揭晓。

```python
1. password_list = ['*#*#','12345']
2.
3. def account_login():
4.     tries = 3
5.     while tries > 0:
6.         password = input('Password:')
7.         password_correct = password == password_list[-1]
8.         password_reset = password == password_list[0]
9.
10.        if password_correct:
11.            print('Login success!')
12.
13.        elif password_reset:
14.            new_password = input('Enter a new password :')
15.            password_list.append(new_password)
16.            print('Password has changed successfully!')
17.            account_login()
18.        else:
19.            print('Wrong password or invalid input!')
20.            tries = tries - 1
21.            print( tries, 'times left')
22.
23.    else:
24.        print('Your account has been suspended')
25.
26.account_login()
```

这段代码只有三处与前面的不一样：

第4~5行：增加了while 循环，如果 tries > 0 这个条件成立，那么便可输入密码，从而执行辨别密码是否正确的逻辑判断；

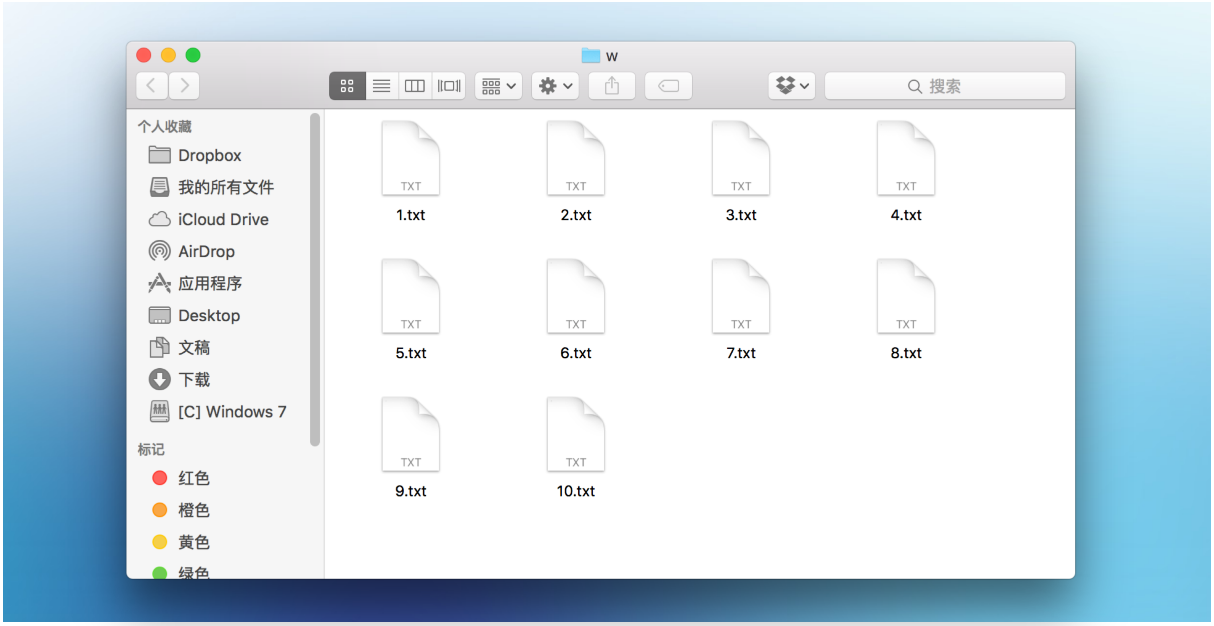
第20~21行：当密码输入错误时，可尝试的次数 tries 减少 1；

第23~24行：while 循环的条件不成立时，就意味着尝试次数用光，通告用户账户被锁。

在这里 while 可以理解成是 if 循环版，可以使用 while-else 结构，而在 while 代码块中又存在着第二层的逻辑判断，这其实构成了嵌套逻辑（Nested Condition）。

练习题

1、设计这样一个函数，在桌面的文件夹上创建10个文本，以数字给它们命名。



2、复利是一件神奇的事情，正如富兰克林所说：“复利是能够将所有铅块变成金块的石头”。设计一个复利计算函数 invest()，它包含三个参数：amount（资金），rate（利率），time（投资时间）。输入每个参数后调用函数，应该返回每一年的资金总额。它看起来就应该像这样（假设利率为5%）：

```
principal amount:100
year 1: $105.0
year 2: $110.25
year 3: $115.7625
year 4: $121.55062500000001
year 5: $127.62815625000002
year 6: $134.00956406250003
year 7: $140.71004226562505
year 8: $147.74554437890632
```

3、打印1～100内的偶数

建议你动手练习一次，然后在微信公众号中回复“**循环**”获得答案，微信公众号是：easypython

综合练习

我们已经基本学完了逻辑判断和循环的用法，现在开始做一点有意思的事情：设计一个小游戏猜大小，这个在文曲星上的小游戏陪伴我度过了小学时的无聊时光。

在此之前，还是先行补充一些必要知识。

首先，创建一个列表，放入数字，再使用 sum() 函数对列表中的所有整数求和，然后打印：

```
a_list = [1,2,3]
print(sum(a_list))
```

结果是6，这应该很好理解。

接着，Python 中最方便的地方是有很多强大的库支持，现在我们导入一个 random 的内置库，然后使用它生成随机数：

```
import random

point1 = random.randrange(1,7)
point2 = random.randrange(1,7)
point3 = random.randrange(1,7)

print(point1,point2,point3)
```

结果就不展示了，因为每次打印结果肯定是不一样的，其中 random 中的 randrange 方法使用起来就像是 range 函数一样，两个参数即可限定随机数范围。

在正式开始创建函数之前，我们先把游戏规则细化一下：

游戏开始，首先玩家选择 Big or Small（押大小），选择完成后开始摇三个骰子计算总值，11 <= 总值 <=18 为“大”，3 <= 总值 <= 10 为“小”。然后告诉玩家猜对或是猜错的结果。看起来就像是这样：

```
<<<<< GAME STARTS! >>>>>
Big or Small:Big
<<<<< ROLE THE DICE!>>>>>
The points are [2, 6, 3] You Lose!
```

好，现在我们就可以开始来制作小游戏了！

我们先来梳理一下这个小游戏的程序设计思路：



摇骰子　　　　转换为大小　　　　猜大小，告知结果

首先，需要让程序知道如何摇骰子，我们需要构建一个摇骰子的函数。这里面有两个关键点，一是需要摇3个骰子，每个骰子都生成1~6的随机数，你需要考虑一下，用什么方式可以实现依次摇3个骰子，这是我们在这一章里面学到的知识点；二是创建一个列表，把摇骰子的结果存储在列表里面，并且每局游戏都更换结果，也就是说每局游戏开始前列表都被清空一次，这里也需要好好考虑下用什么方式实现。

其次，我们摇出来的结果是3个骰子分别的点数，需要把点数转换为“大”或者“小”，其中“大”的点数范围是11 <= 总值 <=18，“小”的点数范围是3 <= 总值 <= 10。

最后，让用户猜大小，如果猜对了就告诉用户赢的结果，如果猜错了就告诉用户输的结果。

只要你掌握了本章的内容，这个小游戏的编程过程并不困难。如果你决心掌握编程这种魔法，实际上最需要的是，发展出设计与分解事物的思路。所谓逻辑关系就是不同事物之间的关联性，它们以何种方式连接、作用，又在什么边界条件下能实现转换或互斥。与其说是编程有趣，倒不如说是编程引发的这种思考给开发者带来了乐趣。

有思路了吗？先试试自己动手做吧。下一页会揭晓答案。

首先，我们先来构造可以摇骰子的函数 roll_dice 。这个函数其实并不需要输入任何参数，调用后会返回储存着摇出来三个点数结果的列表。

```
1. import random
2. def roll_dice(numbers=3, points=None):
3.     print('<<<<< ROLL THE DICE! >>>>>')
4.     if points is None:
5.         points = []
6.     while numbers > 0:
7.         point = random.randrange(1,7)
8.         points.append(point)
9.         numbers = numbers - 1
10.    return points
```

第2行：创建函数，设定两个默认参数作为可选，numbers——骰子数量，points——三个筛子的点数的列表；

第3行：告知用户开始摇骰子；

第4~5行：如果参数中并未指定 points，那么为 points 创建空的列表；

第6~9行：摇三次骰子，每摇一次 numbers 就减 1 ，直至小于等于 0 时，循环停止；

第10行：返回结果的列表。

接着，我们再用一个函数来将点数转化成大小，并使用 if 语句来定义什么是“大”，什么是“小”：

```
1. def roll_result(total):
2.     isBig = 11 <= total <=18
3.     isSmall = 3 <= total <=10
4.     if isBig:
5          return 'Big'
6.     elif isSmall:
7.         return 'Small'
```

第1行：创建函数，其中必要的参数是骰子的总点数；

第2~3行：设定“大”与“小”的判断标准；

第4~7行：在不同的条件下返回不同的结果。

最后，创建一个开始游戏的函数，让用户输入猜大小，并且定义什么是猜对，什么是猜错，并输出对应的输赢结果。

```
1. def start_game():
2.     print('<<<<< GAME STARTS! >>>>>')
3.     choices = ['Big','Small']
4.     your_choice = input('Big or Small :')
5.     if your_choice in choices:
6.         points = roll_dice()
7.         total = sum(points)
8.         youWin = your_choice == roll_result(total)
9.         if youWin:
10.            print('The points are',points,'You win !')
11.        else:
12.            print('The points are',points,'You lose !')
13.    else:
14.        print('Invalid Words')
15.        start_game()
16.start_game()
```

第1行：创建函数，并不需要什么特殊参数；
第2行：告知用户游戏开始；
第3行：规定什么是正确的输入；
第4行：将用户输入的字符串储存在 your_choice中；
第5、13～15行：如果符合输入规范则往下进行，不符合则告知用户并重新开始；
第6行：调用 roll_dice函数，将返回的列表命名为 points；
第7行：点数求和；
第8行：设定胜利的条件——你所选的结果和计算机生成的结果是一致的；
第9~12行：成立则告知胜利，反之，告知失败；
第16行：调用函数，使程序运行。

完成这个小游戏之后，你就可以试着和自己设计的程序玩猜大小了。同时你也掌握了循环和条件判断混用的方法，初步具备了设计更复杂的程序的能力了。

练习题

1、在最后一个项目的基础上增加这样的功能，下注金额和赔率。具体规则如下：

- 初始金额为1000元；
- 金额为0时游戏结束；
- 默认赔率为1倍，也就是说押对了能得相应金额，押错了会输掉相应金额。

```
<<<<<<<<<  GAME STARTS!  >>>>>>>>
Big or Small :Big
How much you wanna bet ? - 1000
<<<<<<<<< ROLL THE DICE! >>>>>>>>>
The points is [6, 1, 5] You Win
You gained 1000,you have 2000 now
<<<<<<<<<  GAME STARTS!  >>>>>>>>
Big or Small :Big
How much you wanna bet ? - 500
<<<<<<<<< ROLL THE DICE! >>>>>>>>>
The points is [4, 1, 1] You Lose
You lost 500,you have 1500 now
<<<<<<<<<  GAME STARTS!  >>>>>>>>
Big or Small :Small
How much you wanna bet ? - 1500
<<<<<<<<< ROLL THE DICE! >>>>>>>>>
The points is [1, 6, 4] You Lose
You lost 1500,you have 0 now
GAME OVER
```

2、我们在注册应用的时候，常常使用手机号作为账户名，在短信验证之前一般都会检验号码的真实性，如果是不存在的号码就不会发送验证码。检验规则如下：

- 长度不少于11位；
- 是移动、联通、电信号段中的一个电话号码；
- 因为是输入号码界面，输入除号码外其他字符的可能性可以忽略；
- 移动号段，联通号段，电信号段如下：

```
CN_mobile = \
[134,135,136,137,138,139,150,151,152,157,158,159,182,183,184,
187,188,147,178,1705]
CN_union = [130,131,132,155,156,185,186,145,176,1709]
CN_telecom = [133,153,180,181,189,177,1700]
```

程序效果如下：

```
Enter Your number :123
Invalid length,your number should be in 11 digits
Enter Your number :12345
Invalid length,your number should be in 11 digits
Enter Your number :11121123123
No such a operator
Enter Your number :13162221340
Operator : China Union
We're sending verification code via text to your phone: 13162221340
```

建议你动手练习一次，然后在微信公众号中回复“**循环与判断提示**”可以获得提示，回复“**循环与判断答案**”可以获得参考答案，微信公众号是：easypython

扫码查看练习题答案

第六章

数据结构

数据结构（Data Structure）

正如在现实世界中一样，直到我们拥有足够多的东西，才迫切需要一个储存东西的容器，这也是我坚持把数据结构放在最后面的原因——直到你掌握足够多的技能，可以创造更多的数据，你才会重视数据结构的作用。这些储存大量数据的容器，在 Python 称之为内置数据结构（Built-in Data Structure）。

我们日常使用的网站、移动应用，甚至是手机短信都依赖于数据结构来进行存储，其中的数据以一种特定的形式储存在数据结构中，在用户需要的时候被拿出来展现。



注：豆瓣电影列表运用的数据结构的展现

Python 有四种数据结构，分别是：列表、字典、元组，集合。每种数据结构都有自己的特点，并且都有着独到的用处。为了避免过早地陷入细枝末节，我们先从整体上来认识一下这四种数据结构：

```
 list = [val1,val2,val3,val4]
 dict = {key1:val1,key2:val2}
tuple = (val1,val2,val3,val4)
  set = {val1,val2,val3,val4}
```

从最容易识别的特征上来说，列表中的元素使用方括号扩起来，字典和集合是花括号，而元组则是圆括号。其中字典中的元素是均带有‘：’的 key 与 value 的对应关系组。

列表（list）

首先我们从列表开始，深入地讲解每一种数据结构。列表具有的最显著的特征是：

1. **列表中的每一个元素都是可变的；**
2. **列表中的元素是有序的，也就是说每一个元素都有一个位置；**
3. **列表可以容纳 Python 中的任何对象。**

列表中的元素是可变的，这意味着我们可以在列表中添加、删除和修改元素。

列表中的每一个元素都对应着一个位置，我们通过输入位置而查询该位置所对应的值，试着输入：

```
Weekday = ['Monday','Tuesday','Wednesday','Thursday','Friday']
print(Weekday[0])
```

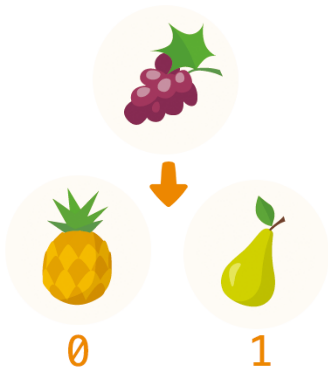
第三个特征是列表可以装入 Python 中所有的对象，看下面的例子就知道了：

```
all_in_list = [
    1,                                  #整数
    1.0,                                #浮点数
    'a word',                           #字符串
    print(1),                           #函数
    True,                               #布尔值
    [1,2],                              #列表中套列表
    (1,2),                              #元组
    {'key':'value'}                     #字典
]
```

列表的增删改查

对于数据的操作，最常见的是增删改查这四类。从列表的插入方法开始，输入：

```
fruit = ['pineapple','pear']
fruit.insert(1,'grape')
print(fruit)
```



在使用 insert 方法的时候，必须指定在列表中要插入新的元素的位置，插入元素的实际位置是在**指定位置元素之前的位置**，如果指定插入的位置在列表中不存在，实际上也就是超出指定列表长度，那么这个元素一定会被放在列表的最后位置。

另外使用这种方法也可以同样达到“插入”的效果：

```
fruit[0:0] = ['Orange']
print(fruit)
```

删除列表中元素的方法是使用 remove()：

```
fruit = ['pinapple','pear','grape']
fruit.remove('grape')
print(fruit)
```

如果要是想替换修改其中的元素可以这样：

```
fruit[0] = 'Grapefruit'
```

删除还有一种方法，那就是使用 del 关键字来声明：

```
del fruit[0:2]
print(fruit)
```

列表的索引与字符串的分片十分相似，同样是分正反两种索引方式，只要输入对应的位置就会返回给你在这个位置上的值：

0	1	2	3	4	5	6	7	8

```
sample = [1,2,3,4,5,6,7,8,9]
```

-9	-8	-7	-6	-5	-4	-3	-2	-1

接下来我们用元素周期表来试验一下：

```
periodic_table = ['H','He','Li','Be','B','C','N','O','F','Ne']
print(periodic_table[0])
print(periodic_table[-2])
print(periodic_table[0:3])
print(periodic_table[-10:-7])
print(periodic_table[-10:])
print(periodic_table[:9])
```

你会发现列表的索引和字符串是一样的，十分简单对吧？但是如果要是反过来，想要查看某个具体的值所在的位置，就需要用别的方法了，否则就会报错：

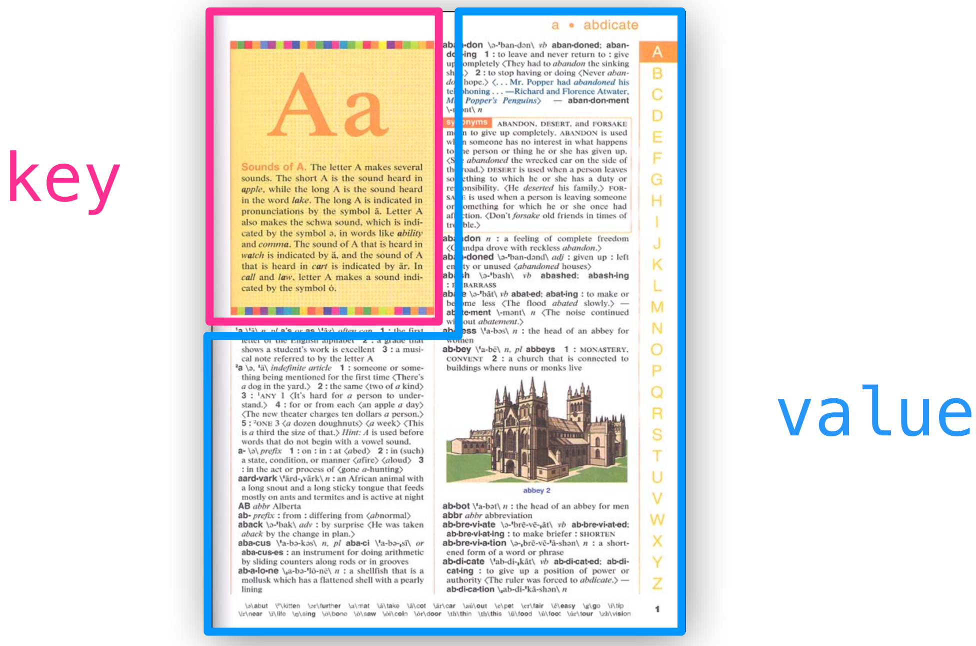
```
print(periodic_table['H'])
```

```
Traceback (most recent call last):
  File "/Users/lingianqian/untitled8/test.py", line 3, in <module>
    print(periodic_table['H'])
TypeError: list indices must be integers or slices, not str
```

报错是因为列表只接受用位置进行索引，但如果数据量很大的话，肯定会记不住什么元素在什么位置，那么有没有一种数据类型可以用人类的方式来进行索引呢？其实这就是字典，我们一起来继续学习。

字典（Dictionary）

编程世界中其实有很多概念都基于现实生活的原型，字典这种数据结构的特征也正如现实世界中的字典一样，使用名称－内容进行数据的构建，在 Python 中分别对应着键（key）－值（value），习惯上称之为键值对。



字典的特征总结如下：

1. **字典中数据必须是以键值对的形式出现的；**
2. **逻辑上讲，键是不能重复的，而值可以重复；**
3. **字典中的键（key）是不可变的，也就是无法修改的；而值（value）是可变的，可修改的，可以是任何对象。**

用下面这个例子来看一下，这是字典的书写方式：

```
NASDAQ_code = {
    'BIDU':'Baidu',
    'SINA':'Sina',
    'YOKU':'Youku'
}
```

一个字典中键与值并不能脱离对方而存在，如果你写成`{'BIDU':}`会引发一个语法错误：

```
    NASDAQ_code = {'BIDU':}
                         ^
SyntaxError: invalid syntax
```

我们再试着将一个可变（mutable）的元素作为 key 来构建字典，比如列表：

```
key_test = {[]:'a Test'}
print(key_test)
```

```
    key_test = {[]:'a Test'}
TypeError: unhashable type: 'list'
```

想必一次次的报错会让你深深记住这两个特征： key 和 value 是一一对应的，key 是不可变的。

同时字典中的键值不会有重复，即便你这么做，相同的键值也只能出现一次：

```
a = {'key':123,'key':123}
print(a)
```

字典的增删改查

首先我们按照映射关系创建一个字典，继续使用前面的例子：

```
NASDAQ_code = {'BIDU':'Baidu','SINA':'Sina'}
```

与列表不同的是，字典并没有一个可以往里面添加单一元素的“方法”，但是我们可以通过这种方式进行添加：

```
NASDAQ_code['YOKU'] = 'Youku'
print(NASDAQ_code)
```

列表中有用来添加多个元素的方法 extend() ，在字典中也有对应的添加多个元素的方法 update()：

```
NASDAQ_code.update({'FB':'Facebook','TSLA':'Tesla'})
```

删除字典中的元素则使用 del 方法：

```
del NASDAQ_code['FB']
```

需要注意的是，虽说字典是使用的花括号，在索引内容的时候仍旧使用的是和列表一样的方括号进行索引，只不过在括号中放入的一定是——字典中的键，也就是说需要通过键来索引值：

```
NASDAQ_code['TSLA']
```

同时，字典是不能够切片的，也就是说下面这样的写法应用在字典上是错误的：

```
chart[1:4]   # WRONG!
```

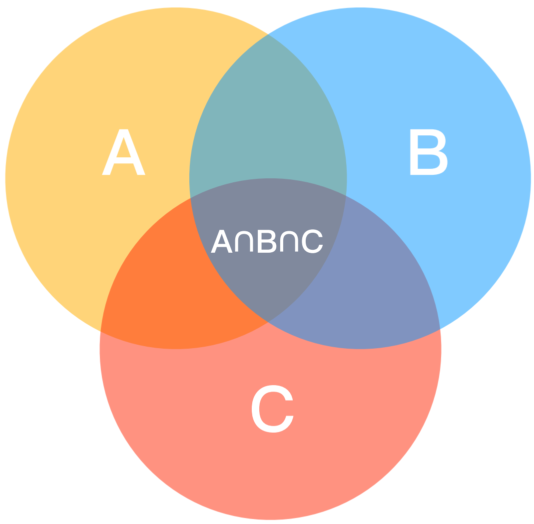
元组（Tuple）

元组其实可以理解成一个稳固版的列表，因为元组是不可修改的，因此在列表中的存在的方法均不可以使用在元组上，但是元组是可以被查看索引的，方式就和列表一样：

```
letters = ('a','b','c','d','e','f','g')
letter[0]
```

集合（Set）

集合则更接近数学上集合的概念。每一个集合中的元素是无序的、不重复的任意对象，我们可以通过集合去判断数据的从属关系，有时还可以通过集合把数据结构中重复的元素减掉。



集合不能被切片也不能被索引，除了做集合运算之外，集合元素可以被添加还有删除：

```
a_set = {1,2,3,4}
a_set.add(5)
a_set.discard(5)
```

数据结构的一些技巧

多重循环

有很多函数的用法和数据结构的使用是息息相关的。前面我们学习了列表的基本用法，而在实际操作中往往会遇到更多的问题。比如，在整理表格或者文件的时候会按照字母或者日期进行排序，在 Python 中也存在类似的功能：

```
num_list = [6,2,7,4,1,3,5]
print(sorted(num_list))
```

怎么样，很神奇吧？sorted 函数按照长短、大小、英文字母的顺序给每个列表中的元素进行排序。这个函数会经常在数据的展示中使用，其中有一个非常重要的地方，sorted 函数并不会改变列表本身，你可以把它理解成是先将列表进行复制，然后在进行顺序的整理。

在使用默认参数 reverse 后列表可以被按照逆序整理：

```
sorted(num_list,reverse=True)
```

在整理列表的过程中，如果同时需要两个列表应该怎么办？这时候就可以用到 zip 函数，比如：

```
for a,b in zip(num,str):
    print(b,'is',a)
```



推导式

现在我们来看数据结构中的推导式（List comprehension），也许你还看到过它的另一种名称叫做列表的解析式，在这里你只需要知道这两个说的其实是一个东西就可以了。

现在我有10个元素要装进列表中，普通的写法是这样的：

```python
a = []
for i in range(1,11):
    a.append(i)
```

下面换成列表解析的方式来写：

```python
b = [i for i in range(1,11)]
```

列表解析式不仅非常方便，并且在执行效率上要远远胜过前者，我们把两种不同的列表操作方式所耗费的时间进行对比，就不难发现其效率的巨大差异：

```python
import time

a = []
t0 = time.clock()
for i in range(1,20000):
    a.append(i)
print(time.clock() - t0, "seconds process time")

t0 = time.clock()
b = [i for i in range(1,20000)]
print(time.clock() - t0, "seconds process time")
```

得到结果：

```
8.999999999998592e-06 seconds process time
0.0012320000000000005 seconds process time
```

列表推导式的用法也很好理解，可以简单地看成两部分。红色虚线后面的是我们熟悉的 for 循环的表达式，而虚线前面的可以认为是我们想要放在列表中的元素，在这个例子中放在列表中的元素即是后面循环的元素本身。

```
list = [item for item in iterable]
```

为了更好地理解这句话，我们继续看几个例子：

```
a = [i**2 for i in range(1,10)]
c = [j+1 for j in range(1,10)]
k = [n for n in range(1,10) if n % 2 ==0]
z = [letter.lower() for letter in 'ABCDEFGHIGKLMN']
```

字典推导式的方式略有不同，主要是因为创建字典必须满足键－值的两个条件才能达成：

```
d = {i:i+1 for i in range(4)}
g = {i:j for i,j in zip(range(1,6),'abcde')}
g = {i:j.upper() for i,j in zip(range(1,6),'abcde')}
```

循环列表时获取元素的索引

现在我们有一个字母表，如何能像图中一样，在索引的时候得到每个元素的具体位置的展示呢？

```
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

```
a is 1
b is 2
c is 3
d is 4
e is 5
f is 6
g is 7
```

前面提到过，列表是有序的，这时候我们可以使用 Python 中独有的函数 enumerate 来进行：

```
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
for num,letter in enumerate(letters):
    print(letter,'is',num + 1)
```

综合项目

为了深入理解列表的使用方法，在本章的最后，我们来做一个词频统计。需要瓦尔登湖的文本，可以在这里下载：http://pan.baidu.com/s/1o75GKZ4，下载后用 PyCharm 打开文本重新保存一次，这是为了避免编码的问题。

之前还是提前做一些准备，学习一些必要的知识。

```
lyric = 'The night begin to shine, the night begin to shine'
words = lyric.split()
```

现在我们使用 split 方法将字符串中的每个单词分开，得到独立的单词：

```
['The', 'night', 'begin', 'to', 'shine']
```

接下来是词频统计，我们使用 count 方法来统计重复出现的单词：

```
path = '/Users/Hou/Desktop/Walden.txt'
with open(path,'r') as text:
    words = text.read().split()
    print(words)
    for word in words:
        print('{}-{} times'.format(word,words.count(word)))
```

结果出来了，但是总感觉有一些奇怪，仔细观察得出结论：

1. 有一些带标点符号的单词被单独统计了次数；
2. 有些单词不止一次地展示了出现的次数；
3. 由于 Python 对大小写敏感，开头大写的单词被单独统计了。

现在我们根据这些点调整一下我们的统计方法，对单词做一些预处理：

```
import string

path = '/Users/Hou/Desktop/Walden.txt'

with open(path,'r') as text:
    words = [raw_word.strip(string.punctuation).lower() for raw_word in text.read().split()]
    words_index = set(words)
    counts_dict = {index:words.count(index) for index in words_index}

for word in sorted(counts_dict,key=lambda x: counts_dict[x],reverse=True):
    print('{} -- {} times'.format(word,counts_dict[word]))
```

第1行：引入了一个新的模块 string 。其实这个模块的用法很简单，我们可以试着把 string.punctuation 打印出来，其实这里面也仅仅是包含了所有的标点符号——!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~。

第4行：在文字的首位去掉了连在一起的标点符号，并把首字母大写的单词转化成小写；

第5行：将列表用set 函数转换成集合，自动去除掉了其中所有重复的元素；

第6行：创建了一个以单词为键（key）出现频率为值（value）的字典；

第7~8行：打印整理后的函数，其中 `key=lambda x: counts_dict[x]` 叫做 lambda 表达式，可以暂且理解为以字典中的值为排序的参数。

第七章

类与可口可乐

All the Cokes are the same and all the Cokes are good.
Liz Taylor knows it,
the President knows it,
the bum knows it,
and you know it.

——Andy Warhol

“在美国，所有人喝到的可乐的都是一样的，无论是总统或者是流浪汉”。波普艺术家 Andy Warhol 如是说。如果用编程的语言来表达 Andy 的思想那么我想可能用**类**（**class**）这个概念最为合适。



```
class CocaCola:
    it_taste = 'So good!'

coke_for_bum = CocaCola()
coke_for_president = CocaCola()

print(coke_for_bum.it_taste)
print(coke_for_president.it_taste)
```

运行结果

```
>>> So good!
>>> So good!
```

运行结果
So good!
good!



怎么样？无论是 Andy Warhol 的波普艺术的理念，还是 Python 中的类的概念，看起来似乎都没有那么难理解。即便你现在对类还充满疑惑，但是我相信，在学习了这一章之后，你将会很好地理解类中那些令人费解的概念。同时，这一章将会使用可乐来讲解关于类的各种概念，实际上编程中的概念都可以在现实生活中找到很好的例子进行对照理解，为了加强理解，你最好现在出门买一瓶可乐，然后开始接下来的学习！

定义一个类

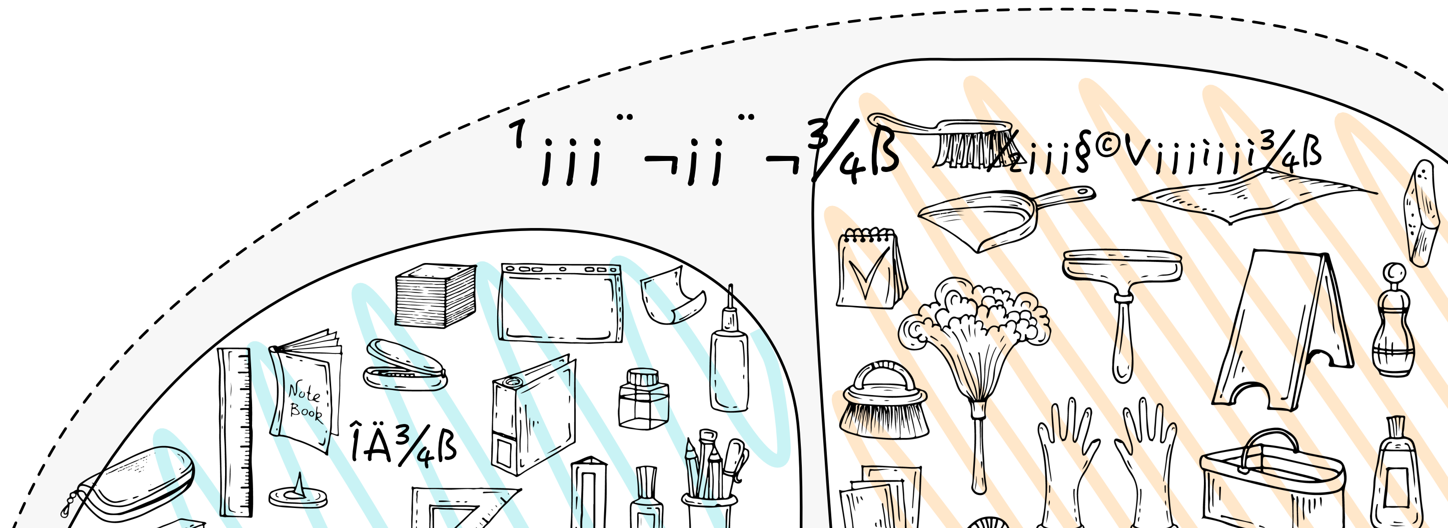
正如“类”的名称一样，它描述的概念和我们现实生活中的类的概念很相似。生物有不同的种类，食物有不同的种类，人类社会的种种商品也有不同的种类。但凡可被称之为一类的物体，他们都有着相似的特征和行为方式。也就是说，**类是有一些系列有共同特征和行为事物的抽象概念的总和。**

对于可乐来讲，只要是同一个品牌的可乐，他们就有着同样的成分，这被称之为配方（formula）。就像是工厂进行批量生产时所遵循的统一标准，正是因为有着同样的配方，所有可口可乐才能达到一样的口味。我们用 Python 中的类来表达这件事：

```
class CocaCola:
    formula = ['caffeine','sugar','water','soda']
```

我们使用 class 来定义一个类，就如同创建函数时使用的 def 定义一个函数一样简单，接着你可以看到缩进的地方有一个装载着列表的变量 `formula`，在类里面赋值的变量就是类的变量，而类的变量有一个专有的术语，我们称之为**类的属性(Class Atrribute)**。

类的变量与我们接触到的变量并没有什么区别，既然字符串、列表、字典、整数等等都可以是变量，那么它们当然都可以成为类的属性，在本章的后面你会逐渐地深入认识这点。



类的实例化

接着我们按照定义好的配方来生产可乐。当然，按照这个配方无论生产多少瓶可乐它们的味道都是一样的。

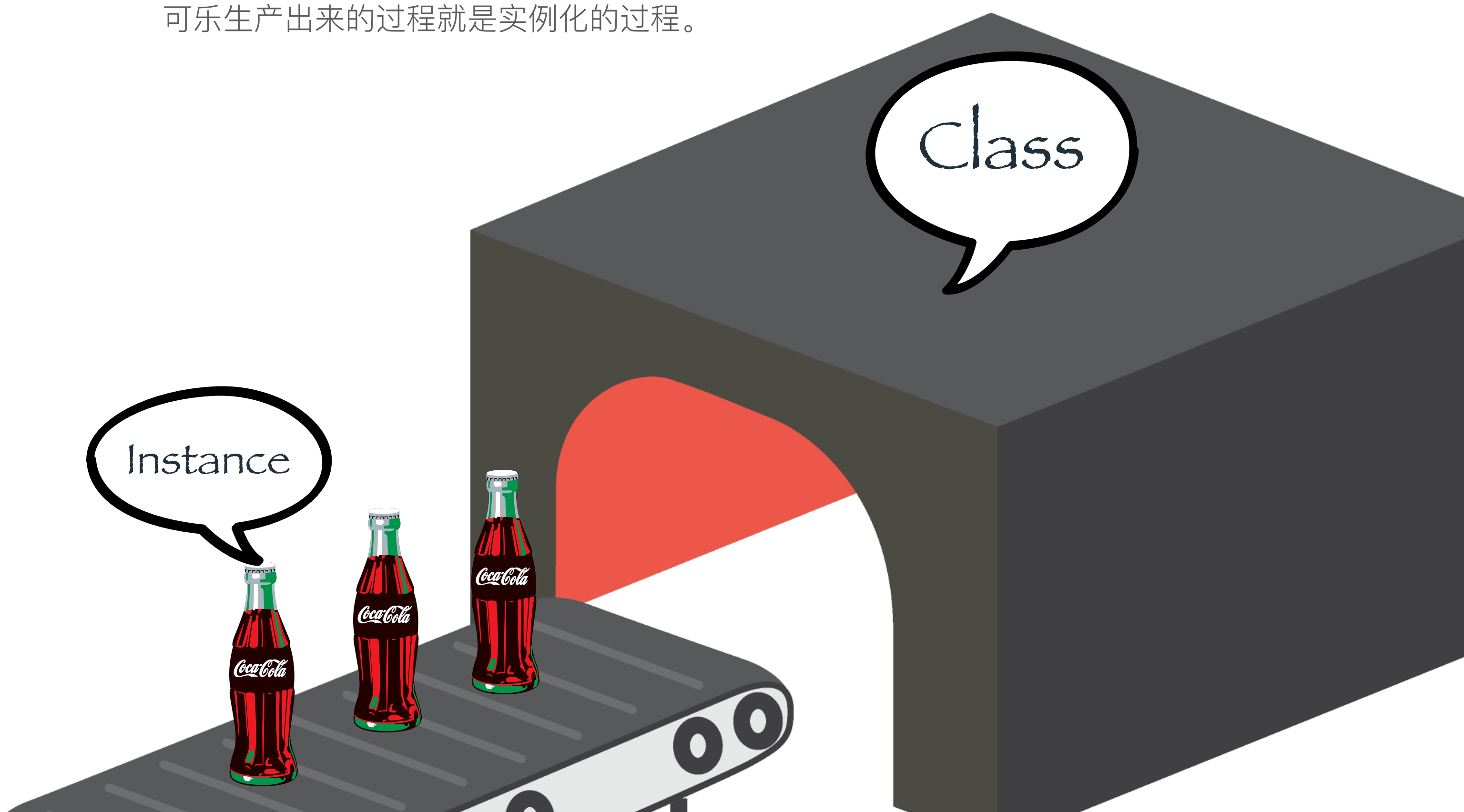
```
coke_for_me = CocaCola()
coke_for_you = CocaCola()
```

```
print(CocaCola.formula)
print(coke_for_me.formula)
print(coke_for_you.formula)
```

```
>>> ['caffeine','sugar','water','soda']
>>> ['caffeine','sugar','water','soda']
>>> ['caffeine','sugar','water','soda']
```

运行结果

在左边我们创建一个变量，右边写上类的名称，这样看起来很像是赋值的行为，我们称之为类的实例化。而被实例化后的对象，我们称之为**实例(instance)**，或者说是类的实例。对于可乐来说，按照配方把可乐生产出来的过程就是实例化的过程。



类属性引用

在类的名字后面输入 `.`，IDE 就会自动联想出我们之前在定义类的时候写在里面的属性，而这就是**类属性的引用(attribute references)**。

类的属性会被所有类的实例共享，所以当你在类的实例后面再点上 `.`，索引用的属性值是完全一样的。

```
print(CocaCola.formula)
print(coke_for_me.formula)
print(coke_for_you.formula)
```

```
>>> ['caffeine','sugar','water','soda']
>>> ['caffeine','sugar','water','soda']
>>> ['caffeine','sugar','water','soda']
```

上面的这几行代码就像是说，“告诉我可口可乐的配方”与“告诉我你手中的可乐的配方”，结果是完全一样的。

类的属性与正常的变量并无区别，你可以试着这样来感受一下：

```
for element in coke_for_me.formula:
    print(element)
```

运行结果：

```
>>> caffeine
>>> sugar
>>> water
>>> soda
```



实例属性

可口可乐风靡全球和其本地化的推广策略有着密不可分的关系。1927年，可口可乐首次进入中国，那时中国人对这个黑色的、甜中带苦的饮料有一种距离感，再加上那时候“可乐”这东西并没有一个官方的翻译，而是直接沿用英文标识“CocaCola”，而民间将这个奇怪的东西称为“蝌蝌啃蜡”。奇怪的味道加上奇怪的名字，导致可乐早期进入中国并没有得到好的反响。

1979年，中国开始大规模开放进出口贸易，官方的、印有我们熟知的“可口可乐”中文标识的可乐才逐渐出现人们的生活中，变得流行起来。同样的配方，不一样的名称，就带来了不同的效果。这说明生产的过程中有必要做一些独有的本地化调整：

```
class CocaCola:
    formula = ['caffeine','sugar','water','soda']

coke_for_China = CocaCola()
coke_for_China.local_logo = '可口可乐' #创建实例属性

print(coke_for_China.local_logo) #打印实例属性引用结果
```

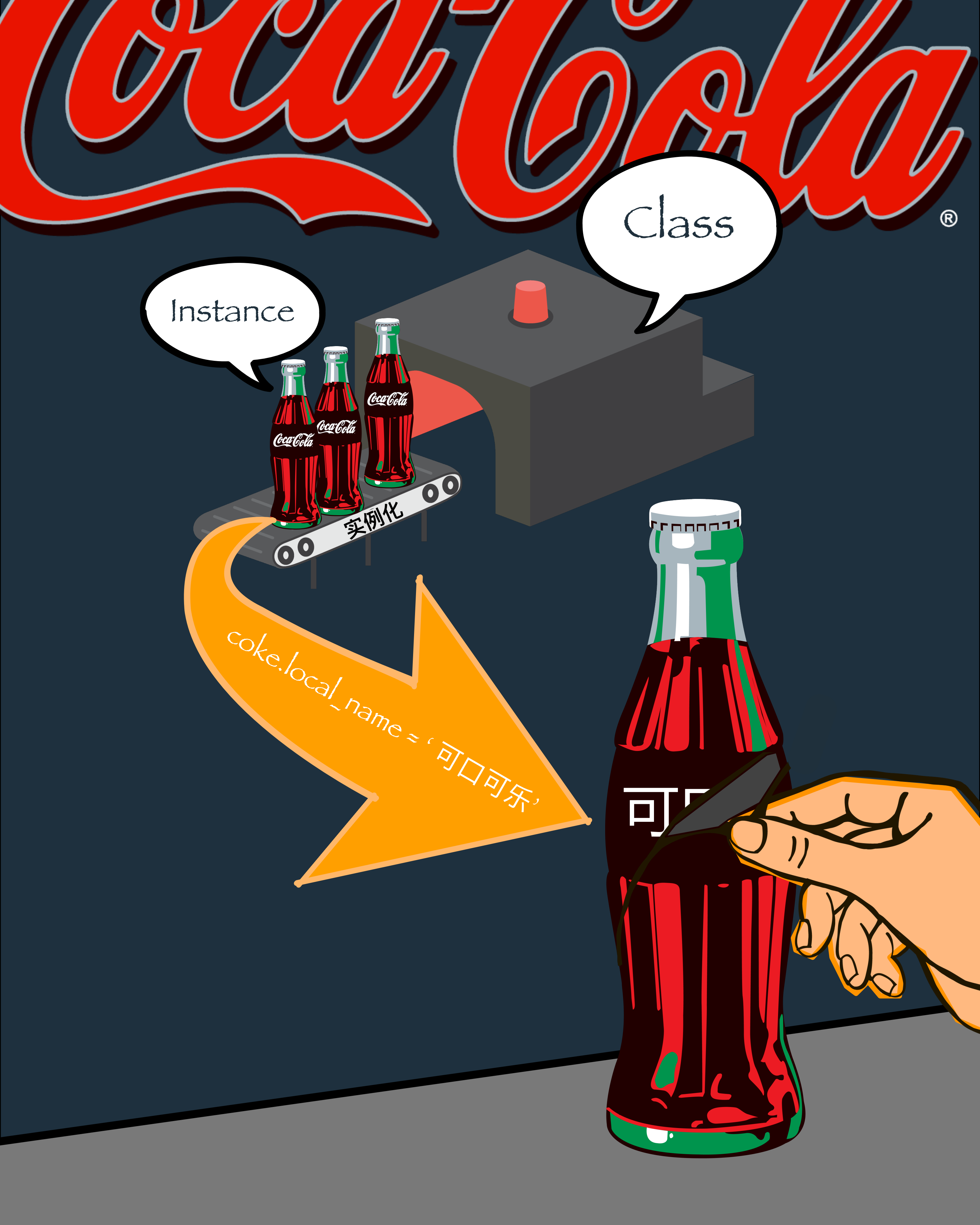




通过上面的代码，我们给在中国生产的可口可乐贴上了中文字样的“可口可乐”标签——在创建了类之后，通过 `object.new_attr` 的形式进行一个赋值，于是我们就得到了一个新的实例的变量，实例的变量就是实例变量，而实例变量有一个专有的术语，我们称之为**实例属性**(Instance Atrribute)。（如果你见过对象属性这种说法，你要知道，这二者其实是在说一件事情。）

可乐的配方 (formula) 属于可口可乐（Class），而“可口可乐”的中文标识(local_logo) 属于中国区的每一瓶可乐（Instance），给中国区的可口可乐贴上中文标签，并不能影响到美国或是日本等其他地区销售的可乐标签。

如果你在这里察觉了一点可疑的、甚至是令人困惑的地方，那么真是一件值得高兴的事情。 因为从引用方式上来说，引用实例属性和引用类属性完全一样！但是这二者却有本质上的差异，后面我们会详细说明类属性和实例属性的区别。


Coca-Cola
Class
Instance
实例化
coke.local_name = '可口可乐'

畅爽开怀，实例方法

类的实例可以引用属性，但我们更早了解到的是类的实例可以使用方法这件事（见第三章：字符串的方法）。方法就是函数，但我们把这个函数称之为**方法(Method)**。方法是供实例使用的，因此我们还可以称之为**实例方法(Instance Method)**。当你掉喝一瓶可乐的时候，你会从咖啡因和大量的糖分中获得能量，如果使用类的方法来表示可乐的这个“功能”的话，那应该是这样的：

```
class CocaCola:
    formula = ['caffeine','sugar','water','soda']
    def drink(self):
        print('Energy!')

coke = CocaCola()
coke.drink()
>>> Energy!
```

BTW：事实上，英文中“功能”和“函数”都由一个词表达—— Function。

self?

我知道你现在的关注点一定在这个奇怪的地方——似乎没有派上任何用场的 `self` 参数。我们来说明一下原理，其实很简单，我们不妨修改一下代码：

```
class CocaCola:
    formula = ['caffeine','sugar','water','soda']
    def drink(coke):    # HERE!
        print('Energy!')

coke = CocaCola()
coke.drink()
>>> Energy!
```

怎么样，现在有些头绪了吧？和你想的一样，这个参数其实就是被创建的实例本身。还记得我们在第四章说的函数的使用办法吗？就是将一个个对象作为参数放入函数括号内。

再进一步说，一旦一个类被实例化，那么我们其实可以使用和与我们使用函数相似的方式：

```
coke = CocaCola
coke.drink() == CocaCola.drink(coke) #左右两边的写法完全一致
```

被实例化的对象会被编译器默默地传入后面方法的括号中，作为第一个参数。上面这两种方法是一样的，但是我们更多地会写成前面那种形式。其实`self`这个参数名称是可以随意修改名称的（编译器并不会因此而报错），但是按照 Python 的规矩，我们还是统一使用`self`。

更多参数

和函数一样，类的方法也能有属于自己的参数，我们先来试着在`.drink()`方法上做些改动：

```
class CocaCola:
    formula = ['caffeine','sugar','water','soda']
    def drink(self,how_much):

        if how_much == 'a sip':
            print('Cool~')
        elif how_much == 'whole bottle'
            print('Headache!')

ice_coke = CocaCola()
ice_coke.drink('a sip')
>>> Cool~
```



“魔术方法”

Python 的类中存在一些方法，被称为"魔术方法”，`__init__()` 就是其中之一。

`__init__()` 的神奇之处就在于，如果你在类里定义了它，在创建实例的时候它就能帮你自动地处理很多事情——比如新增实例属性。在上面的代码中，我们创建了一个实例属性，但那是在定义完类之后再做的，这次我们一步到位：

其实 `__init__()`是initialize(初始化)的缩写，这也就意味着即使我们在创建实例的时候不去引用 init_()方法，其中的命令也会先被自动地执行。是不是感觉像变魔术一样?




```
class CocaCola():
    formula = ['caffeine','sugar','water','soda']
    def __init__(self):
        self.local_logo = '可口可乐'

    def drink(self):    # HERE!
        print('Energy!')

coke = CocaCola()
print(coke.local_logo)
```

`__init__()` 方法可以给类的使用提供了极大的灵活性，试试这段代码会发生什么:

```
class CocaCola:
    formula = ['caffeine','sugar','water','soda']
    def __init__(self):

        for element in self.formula:
            print('Coke has {}!'.format(element))

    def drink(self):
        print('Energy!')

coke = CocaCola()
```

除了必写的self 参数之外，`__init__()` 同样可以有自己的参数，同时也不需要这样 `obj.__init__()` 的方式来调用（因为是自动执行）而是在实例化的时候往类后面的括号中放进参数，相应的所有参数都会传递到这个特殊的 `__init__()` 方法中，和函数的参数的用法完全相同。

```
class CocaCola:
    formula = ['caffeine','sugar','water','soda']
    def __init__(self,logo_name):
        self.local_logo = logo_name

    def drink(self):
        print('Energy!')

coke = CocaCola('可口可乐')
coke.local_logo
>>> 可口可乐
```

如果你对上面代码中的 `self.local_logo = logo_name` 感到不理解，我们在这里可以简单地解释一下。左边是变量作为类的属性，右边是传入的这个参数作为变量，也就是说这个变量的赋值所储存的结果将取决于初始化的时候所传进来的参数 `logo_name`，传进来什么那么它就将是什么。



类的继承

时代在变迁，消费品的种类在不断增长，现在的时代早已经不是 Andy Warhol 那个只有一个口味的可口可乐的时代了，而且也并非是所有可口可乐的味道一样好——如果喝过樱桃味可乐你就一定会明白。可口可乐本身的口味也是根据现代人的需求变了又变。

现在我们使用可口可乐官方网站上最新的配方来重新定义这个类：

> http://www.coca-colaproductfacts.com/en/coca-cola-products/coca-cola/



```python
class CocaCola:
    calories    = 140
    sodium      = 45
    total_carb  = 39
    caffeine    = 34
    ingredients =  [

        'High Fructose Corn Syrup',
        'Carbonated Water',
        'Phosphoric Acid',
        'Natural Flavors',
        'Caramel Color',
        'Caffeine'
    ]
    def __init__(self,logo_name):
        self.local_logo = logo_name

    def drink(self):
        print('You got {} cal energy!'.format(self.calories))
```

不同的本地化策略和新的种类的开发，使得生产并非仅仅是换个标签这么简单了，包装、容积、甚至是配方都会发生变化，但唯一不变的是：它们永远是可口可乐。

所有的子品类都会继承可口可乐的品牌，Python 中类自然也有对应的概念，叫做**类的继承 (Inheritance)**，我们拿无咖可乐（CAFFEINE-FREE）作为例子：

```
class CaffeineFree(CocaCola):
    caffeine = 0
    ingredients =  [

        'High Fructose Corn Syrup',
        'Carbonated Water',
        'Phosphoric Acid',
        'Natural Flavors',
        'Caramel Color',
    ]

coke_a = CaffeineFree('Cocacola-FREE')

coke_a.drink()
```

我们在新的类 CaffeineFree 后面的括号中放入 CocaCola，这就表示这个类是继承于 CocaCola 这个父类的，而 CaffeineFree 则成为了 CocaCola 子类。类中的变量和方法可以完全被子类继承，但如需有特殊的改动也可以进行**覆盖（Override）**。

可以看到CAFFEINE-FREE存在着咖啡因含量、成分这两处不同的地方，并且在新的类中也仅仅是重写了这两个地方，其他没有重写的地方，方法和属性都能照常使用。



令人困惑的类属性与实例属性

Q1

类属性如果被重新赋值，是否会影响到类属性的引用?

```python
class TestA:
    attr = 1
obj_a = TestA()

TestA.attr = 42
print(obj_a.attr)
```

Q2

实例属性如果被重新赋值，是否会影响到类属性的引用?

```python
class TestA:
    attr = 1
obj_a = TestA()
obj_b = TestA()

obj_a.attr = 42

print(obj_b.attr)
```

Q3

类属性实例属性具有相同的名称，那么 `.` 后面引用的将会是什么?

```python
class TestA:
    attr = 1
    def __init__(self):
        self.attr = 42

obj_a = TestA()

print(obj_b.attr)
```

也许运行完上面三段代码，你会有一些初步的结论，但是更为直接的解释，全部隐藏在类的特殊属性 `__dict__` 中。`__dict__` 是一个类的特殊属性，它是一个字典，用于储存类或者实例的属性。即使你不去定义它，它也会存在于每一个类中，是默认隐藏的。我们以问题3中的代码为背景，在下面添加上黄圈中这两行：

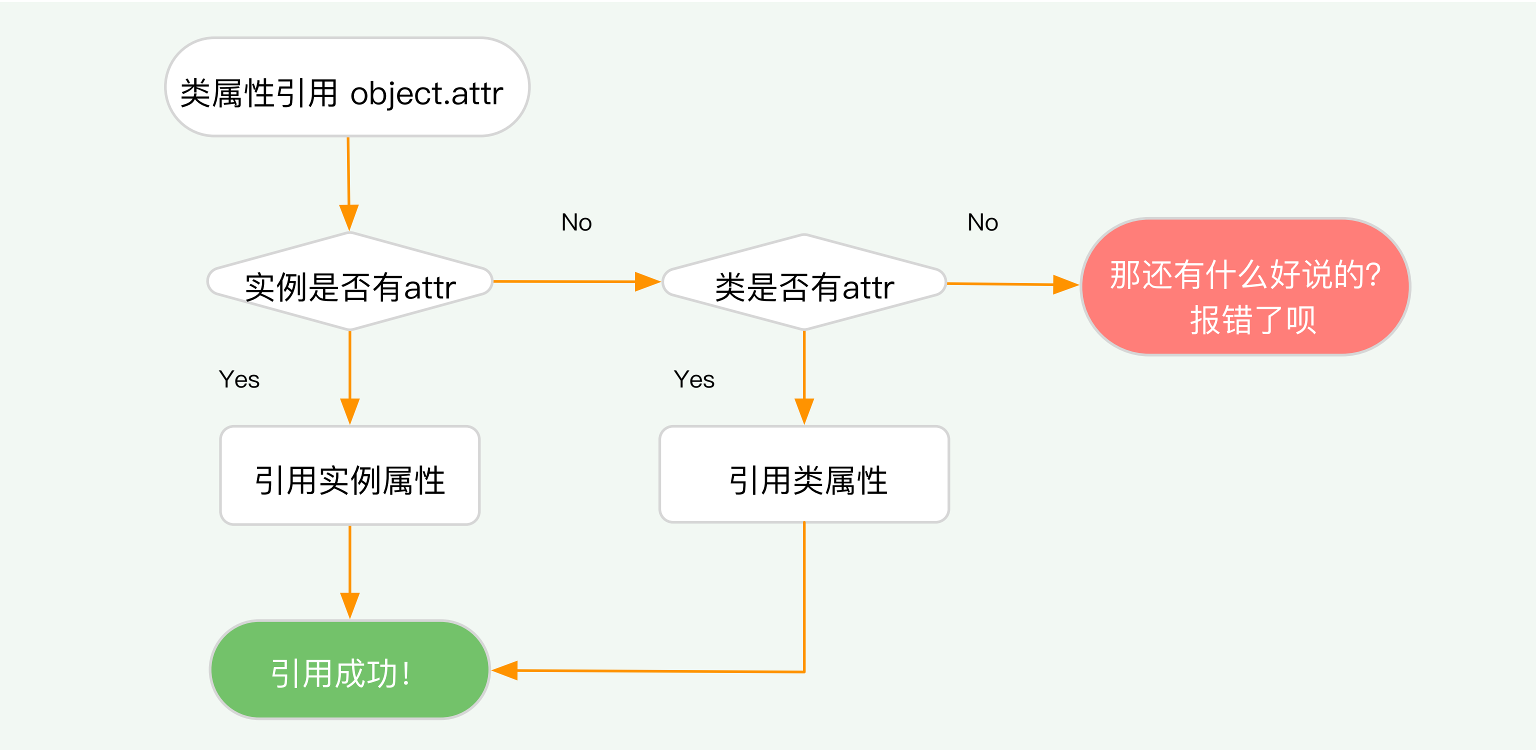
其中类 `TestA` 和类的实例 `obj_a` 各自拥有各自的 attr 属性，井水不犯河水，是完全独立的。但是这样一来又如何解释问题1中的情况呢？为什么引用实例属性的方式会和引用类属性的方式是一样的呢？为何在问题3中返回的是42而不是1呢？请看后面的图：

```
print(TestA.__dict__)
print(obj_a.__dict__)
```

```
>>> {‘__module__': '__main__', '__doc__':
None, '__dict__': <attribute '__dict__' of
'TestA' objects>, '__init__': <function
TestA.__init__ at 0x1007fc7b8>, 'attr': 1,
'__weakref__': <attribute '__weakref__' of
'TestA' objects>}

>>> {'attr': 42}
```

如图所示， Python 中属性的引用机制是自外而内的，当你创建了一个实例之后，准备开始引用属性，这时候编译器会先搜索该实例是否拥有该属性，如果有，则引用；如果没有，将搜索这个实例所属的类是否有这个属性，如果有，则引用，没有那就只能报错了。



类的扩展理解

现在试着敲下这几行代码:

```
obj1 = 1
obj2 = 'String!'
obj2 = []
obj4 = {}

print(type(obj1),type(obj2),type(obj3),type(obj4))
```

Python 中任何种类的对象都是类的实例，上面的这些类型被称作内建类型，它们并不需要像我们上面一样实例化。

如果你安装了 Beautifulsoup4 这个第三方的库，你可以试着这样:

```
from bs4 import Beautifulsoup
soup = BeautifulSoup
print(type(soup))
```

然后你可以按住 cmd （win系统为ctr）点击 Beautifulsoup 来查看一个soup 对象的完整类定义。

到了这里你已经掌握类的基础用法，现在我们还不想把事情搞得复杂，至少现在还不值得浪费更多时间去深入你暂时不会使用到的高级概念。**要记住，不是越多就越好，你不可能在短时间内掌握诸多交织密集的抽象概念。**（一些关于类的高级概念你会在后面的实战项目中学到）

你不用担心如何验证自己是否掌握类的概念，接下来将带你做一个关于类的实践，到那时你可以验证自己的理解是否到位。

类的实践

其实类背后所承载的理念是 OOP（面向对象）的编程理念。在大型项目中为了易于管理和维护代码质量，会采取面向对象的方式，这也是软件工程的智慧所在。接下来，我们将使用类的概念来编写一个日常的工具库导入到 Python 的库中，这样一来我们也可以使用 import 方法导入自己的库了。

在使用 Python 处理数据或者是开发网站时，有时候会使用一些无意义的假数据，比如用户详情页信息。

我们来制作这样一个填充用户假数据的小工具，这个小工具的设计思路如下：

父类：FakeUser
功能：
1. 随机姓名
    a. 单字名
    b. 双字名
    c. 无所谓是什么反正有个名字就好了
2. 随机性别

子类：SnsUser
功能：
1. 随机数量的跟随者
    a. few
    b. a lot

开始之前先来处理一下词库，我们使用的随机姓名的词库来自于某输入法的姓名词库解析后的结果，现在分成两个文件，一个是常见姓氏，一个是常见的名。使用 open 函数打开这两个文件，将其中的文字添加进列表中。我们获取全部的常见姓氏，后面的姓名只获取5000个即可，否则太占内存。

```
ln_path = '/Users/Hou/Desktop/last_name.txt'
fn_path = '/Users/Hou/Desktop/first_name.txt'
fn  = []
ln1 = [] #单字名
ln2 = [] #双字名
with open(fn_path,'r') as f:
    for line in f.readlines():
        fn.append(line.split('\n')[0]) #如果这里看不明白不妨试试对其中的一行使用 split 方法看看会返回回来什么结果
print(fn)
with open(ln_path,'r') as f:
    for line in f.readlines():
        if len(line.split('\n')[0]) == 1:
            ln1.append(line.split('\n')[0])
        else:
            ln2.append(line.split('\n')[0])
print(ln1)
print('='*70) #分割线
print(ln2)
```

打印完成后，我们做两件事情：

1. 将 fn = []、ln1 = []、ln2 = [] 修改改成元组，元组比列表要更省内存。
2. 将打印出来的结果复制粘贴到元组中，显然在制作完成后我们不能每做一次就重新读一遍，要把这些变成常量。

完成后看起来应该像这样（当然比这个要长很多很多）：

```
fn = ('李', '王', '张', '刘')
ln1 = ('娉', '览', '莱', '屹')
ln2 = ('治明', '正顺', '书铎'））
```

现在开始我们可以来定义父类 FakeUser 了：

```
import random
class FakeUser:
    def fake_name(self,one_word=False,two_words=False):
        if one_word:
            full_name = random.choice(fn) + random.choice(ln1)
        elif two_words:
            full_name = random.choice(fn) + random.choice(ln2)
        else:
            full_name = random.choice(fn) + random.choice(ln1
+ ln2)
        print(full_name)
    def fake_gender(self):
        gender = random.choice(['男','女','未知'])
        print(gender)
```

接下来是定义子类：

```
class SnsUser(FakeUser):
    def get_followers(self,few=True,a_lot=False):
        if few:
            followers = random.randrange(1,50)
        elif a_lot:
            followers = random.randrange(200,10000)
    print(followers)
user_a = FakeUser()
user_b = SnsUser()
user_a.fake_name()
user_b.get_followers(few=True)
```

似乎这并没有解决什么问题?

到了这里，我们创建的类已经可以正常使用了。我们的目的是批量制造假的填充数据，但是这样使用比起手工添加，效果并好不到哪去，因此在原有的代码上，我们要做一些小的调整，把所有的 print 替换成 yield 并在其上方加上一层循环，然后神奇的事情就这样发生了，我们就可以像 range 函数一样使用方法了：

```
class FakeUser():
    def fake_name(self,amount=1,one_word=False,two_words=False):
        n = 0
        while n <= amount:
            if one_word:
                full_name = random.choice(fn) + random.choice(ln1)
            elif two_words:
                full_name = random.choice(fn) + random.choice(ln2)
            else:
                full_name = random.choice(fn) + random.choice(ln1 + ln2)
            yield full_name
            n+=1
    def fake_gender(self,amount=1):
        n = 0
        while n <= amount:
            gender = random.choice(['男','女','未知'])
            yield gender
            n +=1
class SnsUser(FakeUser):
    def get_followers(self,amount=1,few=True,a_lot=False):
        n = 0
        while n <= amount :
            if few:
                followers = random.randrange(1,50)
            elif a_lot:
                followers = random.randrange(200,10000)
            yield followers
            n+=1
```

```
user_a = FakeUser()
user_b = SnsUser()
for name in user_a.fake_names(30):
    print(name)
for gender in user_a.fake_gender(30):
    print(gender)
```

为什么?

这其实用到了一个新的概念，叫做**生成器**（generator）。简单的来说，在函数中我们只要在任意一种循环（loop）中使用 yield 返回结果，就可以得到类似于 range 函数的效果。

安装自己的库

我们一般使用 pip 来进行第三方库的安装，那么自己的库要怎么安装呢？当然可以把自己的库提交到 pip 上，但是还要添加一定量的代码和必要文件才行。在这里我们使用一个更简单的方法：

1. 找到你的 Python 安装目录，找到下面的 site-packages 文件夹；
2. 记住你的文件名,因为它将作为引用时的名称，然后将你写的 py 文件放进去。

这个文件夹应该有你所装的所有第三方库。

如果你并不清楚你的安装路径，你可以尝试使用如下方式搞清楚它究竟在哪里：

```
import sys
print(sys.path)
```

打印出来的会是一个列表，列表中的第四个将是你的库安装路径所在，因此你也可以直接这么做：

```
import sys
print(sys.path[3])
```

现在就来试试使用自己写的库吧！

第八章

开始使用第三方库

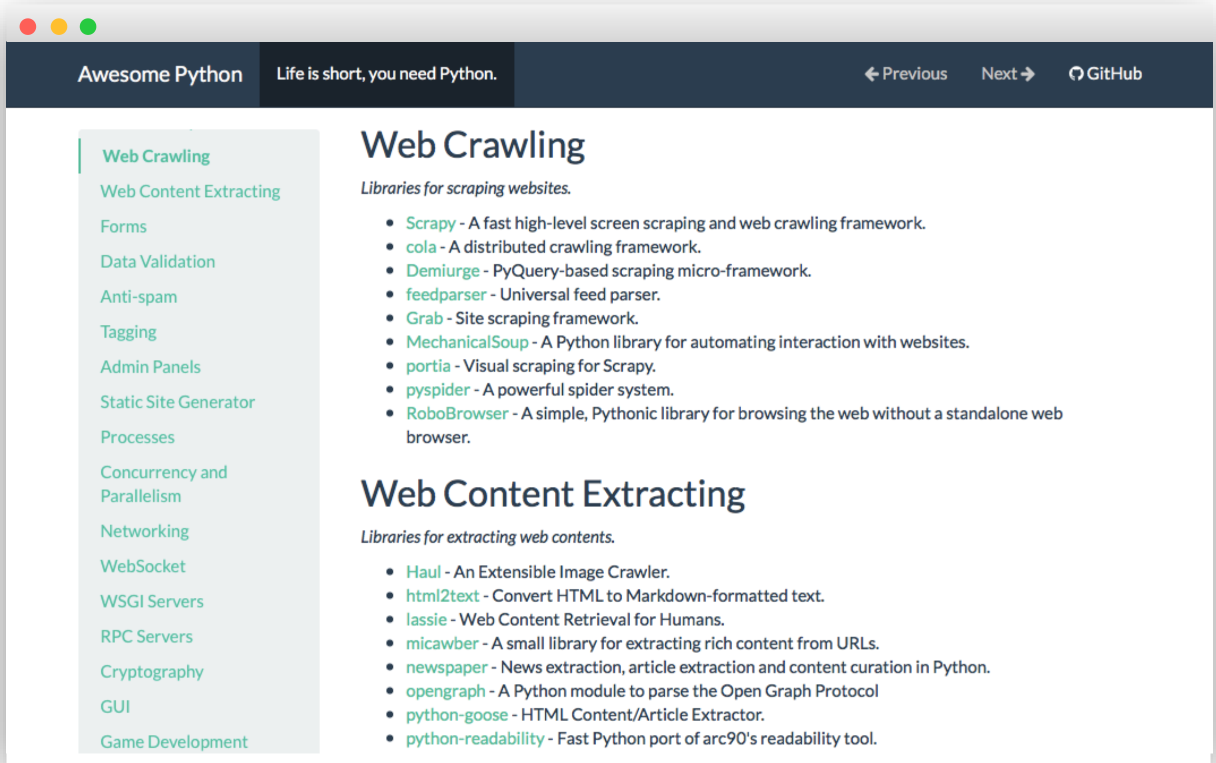
令人惊叹的第三方库

如果用手机来比喻编程语言，那么 Python 是一款智能机。正如海量的手机应用出现在 iOS、Android 平台上，同样有各种各样的第三方库为 Python 开发者提供了极大的便利。

当你想搭建网站时，可以选择功能全面的 Django、轻量的 Flask 等 web 框架；当你想写一个小游戏的时候，可以使用 PyGame 框架；当你想做一个爬虫时，可以使用 Scrapy 框架；当你想做数据统计分析时，可以使用 Pandas 数据框架……这么多丰富的资源可以帮助我们高效快捷地做到想做的事，就不需要再重新造轮子了。

那么，如何根据自己的需求找到相应的库呢？

可以在 awesome-python.com 这个网站上按照分类去寻找，上面收录了比较全面的第三方库。比如当我们想找爬虫方面的库时，查看 Web Crawling 这个分类，就能看到相应的第三方库的网站与简介：

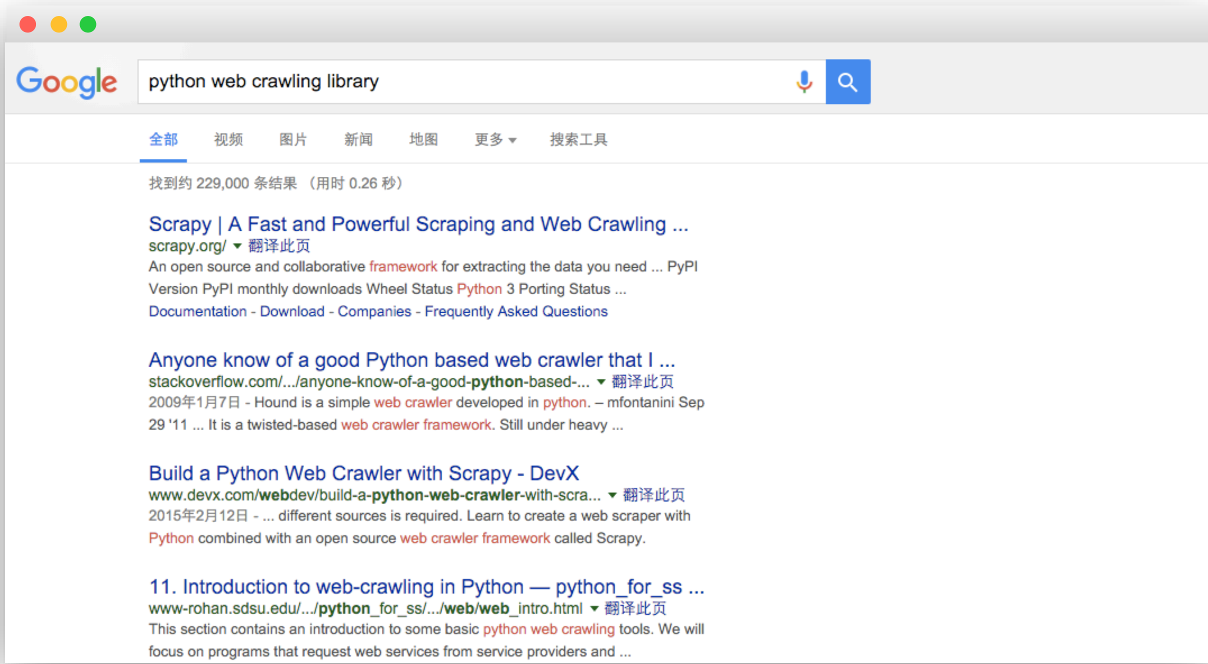


可以进入库的网站查看更详细的介绍，并确认这个库支持的是 python 2 还是 python 3，不过绝大多数常用库已经都支持了这两者。

另外，还可以直接通过搜索引擎寻找，比如：



如果你能尝试用英文搜索，会发现更大的世界，比如 stackoverflow 上的优质讨论。



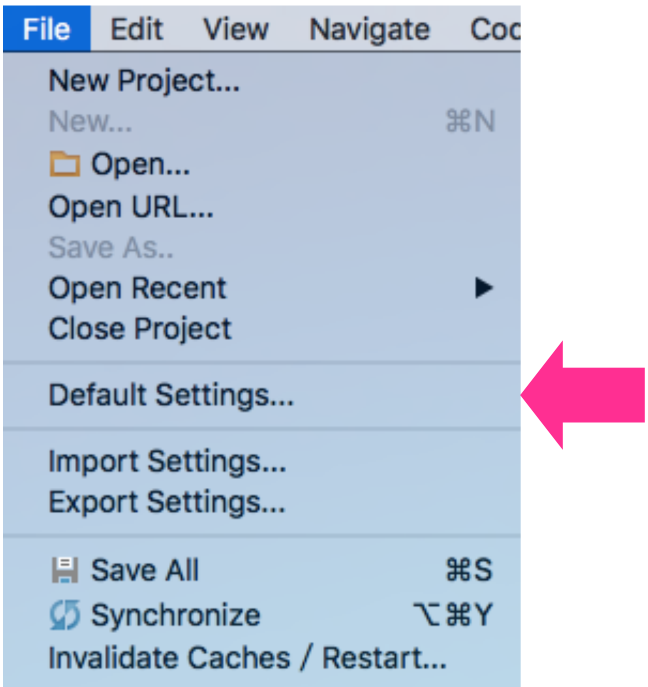
安装第三方库

无论你想安装哪一种库，方法基本都是通用的。下面开始介绍安装第三方库的方法。

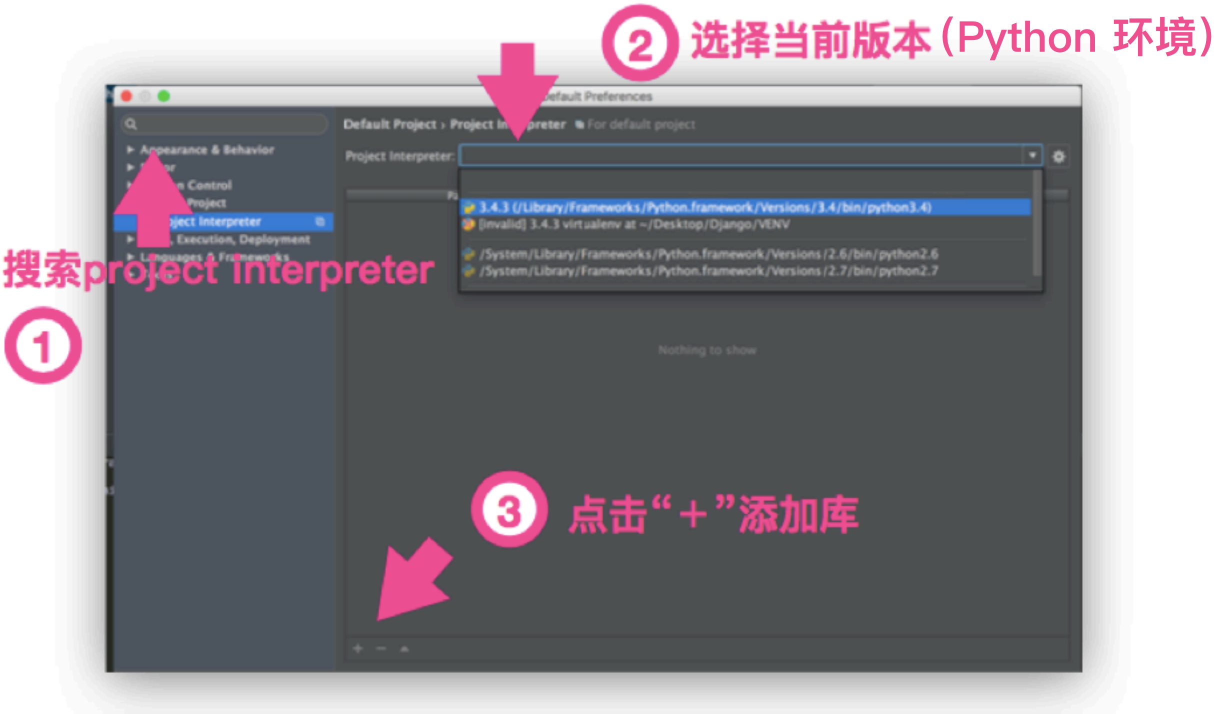
最简单的方式：在 PyCharm 中安装

推荐大家使用 PyCharm ，就是因为它贴心地考虑了开发者的使用体验，在 PyCharm 中可以方便快捷地安装和管理库。

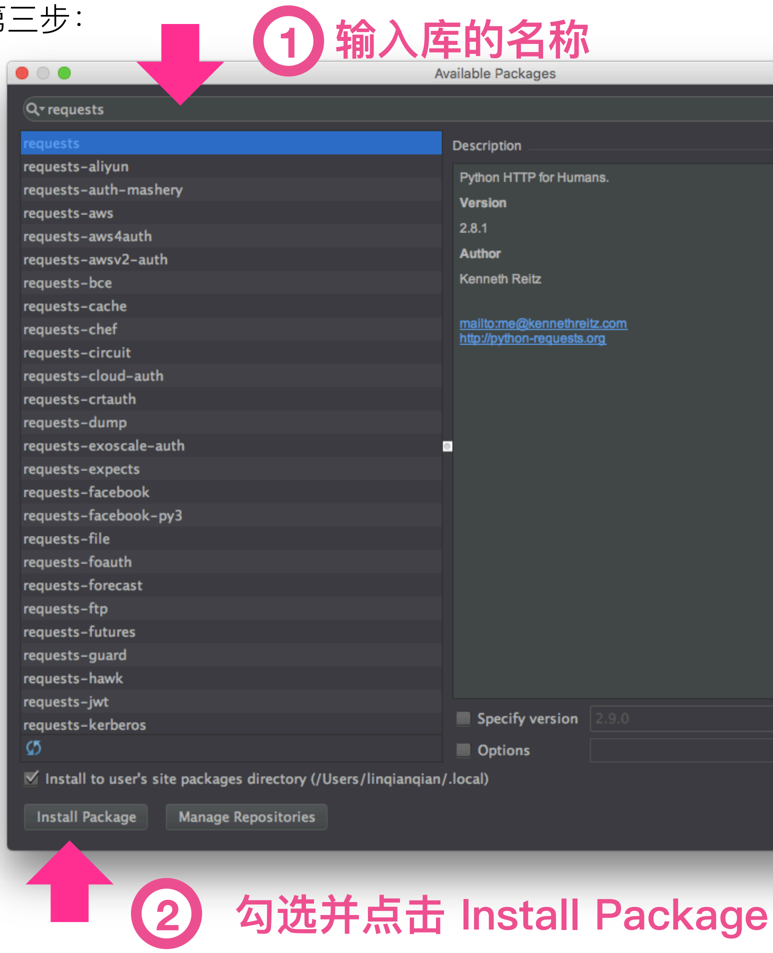
- 第一步：在 PyCharm 的菜单中选择：File > Default Settings



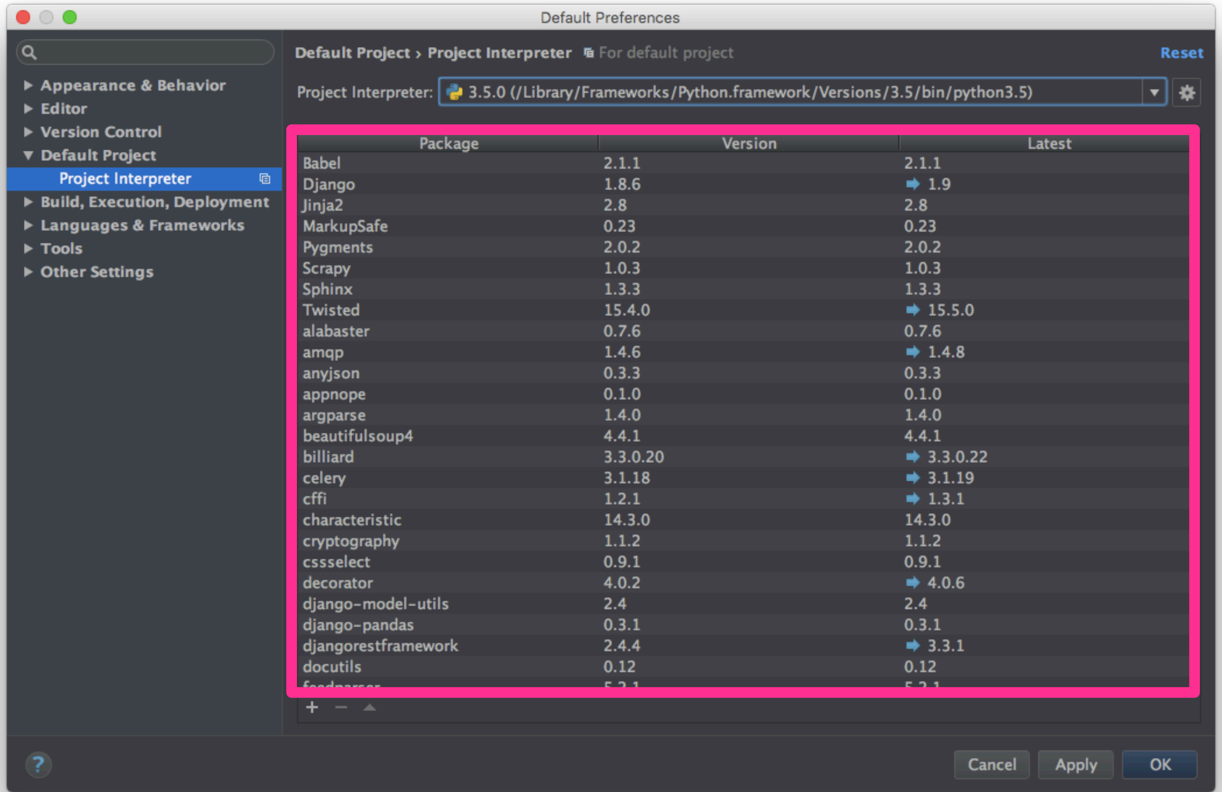
- 第二步：


② 选择当前版本（Python 环境）
搜索project interpreter
①
③ 点击“+”添加库
Project Interpreter:
3.4.3 (/Library/Frameworks/Python.framework/Versions/3.4/bin/python3.4)
[invalid] 3.4.3 virtualenv at ~/Desktop/Django/VENV
/System/Library/Frameworks/Python.framework/Versions/2.6/bin/python2.6
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7
Nothing to show

- 第三步：


① 输入库的名称
Available Packages
requests
requests
requests-aliyun
requests-auth-mashery
requests-aws
requests-aws4auth
requests-awsv2-auth
requests-bce
requests-cache
requests-chef
requests-circuit
requests-cloud-auth
requests-crtauth
requests-dump
requests-exoscale-auth
requests-expects
requests-facebook
requests-facebook-py3
requests-file
requests-foauth
requests-forecast
requests-ftp
requests-futures
requests-guard
requests-hawk
requests-jwt
requests-kerberos
Description
Python HTTP for Humans.
Version
2.8.1
Author
Kenneth Reitz
mailto:me@kennethreitz.com
http://python-requests.org
Specify version
2.9.0
Options
Install to user's site packages directory (/Users/linqianqian/.local)
Install Package
Manage Repositories
② 勾选并点击 Install Package

在安装成功后，PyCharm 会有成功提示。你也可以在 project interpreter 这个界面中查看安装了哪些库，点–号就可以卸载不再需要的库。



最直接的方式：在终端/命令行中安装

安装 pip

在 Python 3.4 之后，安装好 Python 环境就可以直接支持 pip，你可以在终端/命令行里输入这句检查一下：

```
pip --version
```

如果显示了 pip 的版本，就说明 pip 已经成功安装了。
如果发现没有安装 pip的话,各系统安装的方法不同：

- Windows用户请查看：https://taizilongxu.gitbooks.io/stackoverflow-about-python/content/8/README.html
- Mac 用户请查看：https://www.mobibrw.com/p=1274
- Linux 用户请查看：http://pip-cn.readthedocs.org/en/latest/installing.html

使用 pip 安装库

在安装好 pip 之后，以后安装库，只需要在命令行里面输入：
（如果你想安装到 python 2中，需要把 pip3换成 pip ）

```
pip3 install PackageName
```

如果你安装了 python 2和3两种版本，可能会遇到安装目录的问题，可以换成:
（如果你想安装到 python 2中，需要把 python3换成 python ）

```
python3 -m pip install PackageName
```

如果遇到权限问题，可以输入：

```
sudo pip install PackageName
```

安装成功后会提示：

```
Successfully installed PackageName
```

再介绍几个pip 的常用指令：

```
pip install --upgrade pip                    #升级 pip
pip uninstall flask                          #卸载库
pip list                                     #查看已安装库
```

异常情况：安装某些库的时候，可能会遇到所依赖的另一个库还没安装，导致无法安装成功的情况，这时候的处理原则就是：缺啥装啥，举个例子：

```
danbao$ scrapy version -v
:0: UserWarning: You do not have a working installation of the
service_identity module: 'No module named service_identity'.
Please install it from <https://pypi.python.org/pypi/
service_identity> and make sure all of its dependencies are
satisfied.  Without the service_identity module and a recent
enough pyOpenSSL to support it, Twisted can perform only
rudimentary TLS client hostname verification.  Many valid
certificate/hostname mappings may be rejected.
Scrapy  : 0.24.6
lxml    : 3.4.4.0
libxml2 : 2.9.0
Twisted : 15.2.1
Python  : 2.7.9 (default, May 27 2015, 22:47:13) - [GCC 4.2.1
Compatible Apple LLVM 6.1.0 (clang-602.0.53)]
```

这时候的解决方法是：

```
pip install service_identity
```

最原始的方式：手动安装

为了应对异常情况，再提供一种最原始的方法：手动安装。往往是 Windows 用户需要用到这种方法。

进入pypi.python.org，搜索你要安装的库的名字，这时候有3种可能，

- 第一种是 exe 文件，这种最方便，下载满足你的电脑系统和 python 环境的对应的exe，再一路点击 next就可以安装。
- 第二种是 .whl 类文件，好处在于可以自动安装依赖的包。
- 第三种是源码，大概都是 zip 、 tar.zip、 tar.bz2格式的压缩包，这个方法要求用户已经安装了这个包所依赖的其他包。 例如pandas依赖于numpy, 你如果不安装numpy, 这个方法是无法成功安装pandas的。如果没有前两种类型的文件，那只能用这个了。

1、如果你选择了下载.whl 类文件，下面是安装方法：

1）到命令行输入：

```
pip3 install wheel
```

等待执行完成，不能报错。（python2要换成 pip）

2）从资源管理器中确认你下载的.whl 类文件的路径，然后在命令行继续输入：

```
cd C:\download
```

此处需要改为你的路径，路径的含义是文件所在的文件夹，不包含这个文件名字本身

然后再在命令行继续输入：

```
pip3 install xxx.whl
```

xxx.whl 是你下载的文件的完整文件名。

2、如果你选择了下载源码压缩包，下面是安装方法：

1）解压包，进入解压好的文件夹，通常会看见一个 setup.py 的文件。从资源管理器中确认你下载的文件的路径，打开命令行（cmd），输入：

```
cd C:\download
```

此处需要改为你的路径，路径的含义是文件所在的文件夹，不包含这个文件名字本身

2）然后在命令行中继续输入：

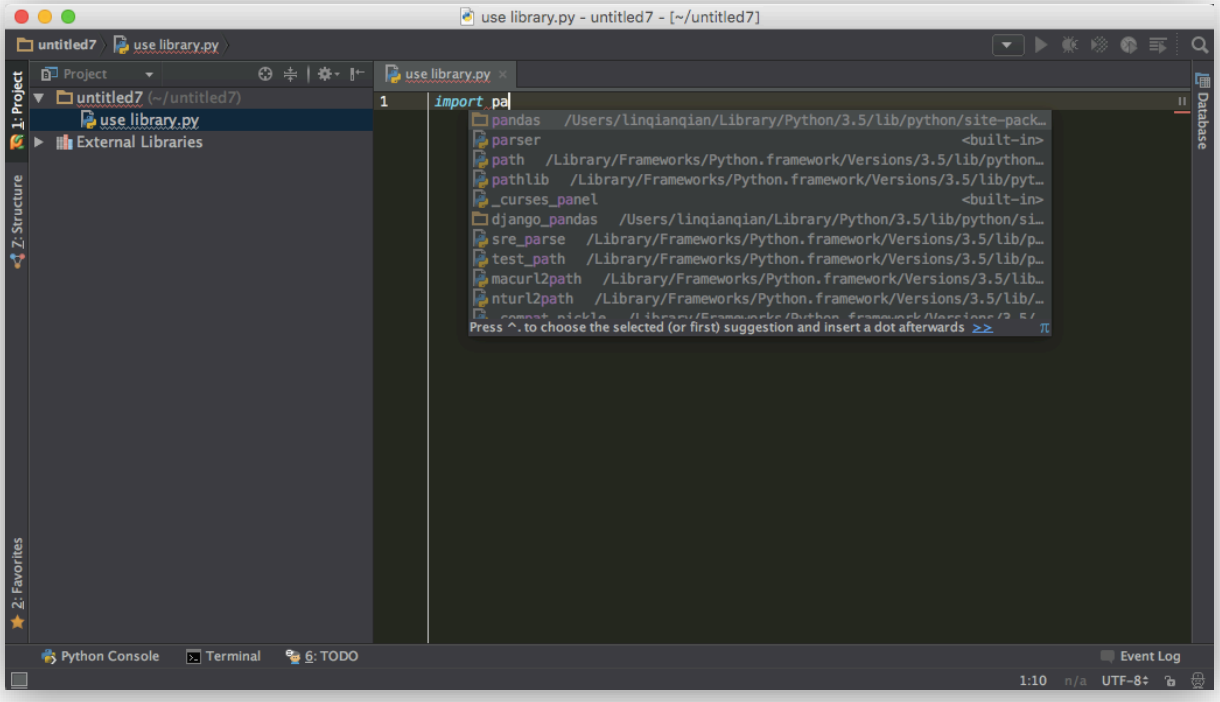
```
python3 setup.py install
```

这个命令，就能把这个第三库安装到系统里，也就是你的 Python 路径，windows大概是在 C:\Python3.5（或2.7）\Lib\site-packages。

想卸载库的时候，找到 python 路径，进入site-packages文件夹，在里面删掉库文件就可以了。

使用第三方库

在 PyCharm 中输入库的名字，就会自动提示补全了：



输入之后你会发现是灰色的状态`import pandas`，这是因为还没有在程序中使用这个库，而不是因为库还没安装（想检查库是否安装的话，可以使用前面提到的 PyCharm 或者 pip 的方式来确认）。

关于库的使用指南到这里就结束了。

必读

给编程小白的学习资源

必读：给编程小白的学习资源

这本书到这里就告一段落了，相信你已经迫不及待地想要找点项目练练手了。在这里推荐一些精选的学习资源，让你持续提升：

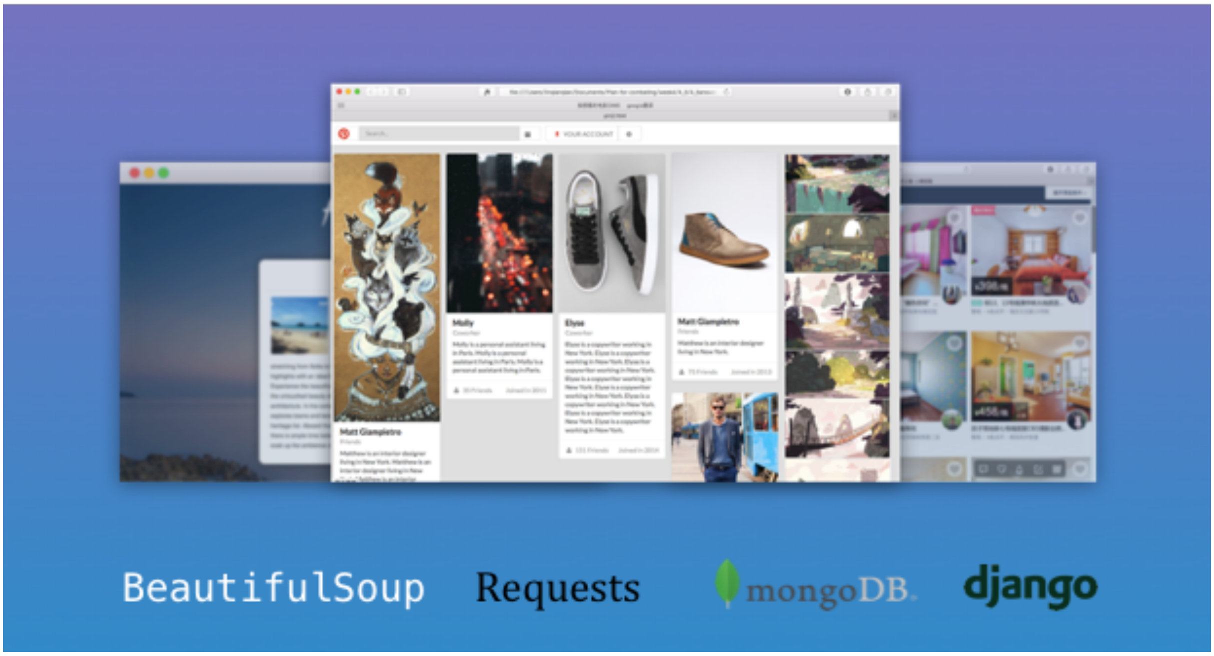
练手项目

在前言也说过，通过项目实践去提升是最快的，毕竟编程是一个 learning by doing 的技能。但是新手又不适合直接做难度很高的项目，最好的实践方式是：分解练习+循序渐进。其实这和学习吉他很像，分解练习能让你对每一个知识点都熟练运用，循序渐进则能让你的能力随着任务难度不断提升。但是，这种实践方式需要被精心设计，作为缺乏经验的新手可能很难制定出这样的学习计划，推荐你选择精心设计的、以练手项目为主的实践课程。

1.麻瓜编程的 Python 实战计划课程

Python 实战：四周实现爬虫网站

http://study.163.com/course/introduction.htm?courseId=1002794001



如果你喜欢这本书，相信你也会喜欢这门 Python 实战课程，这也是我们团队的作品，延续了这本书的风格，生动、易懂、好看。有零基础学员说：“这是我唯一能听懂的编程课，和看美剧似的，会上瘾。”

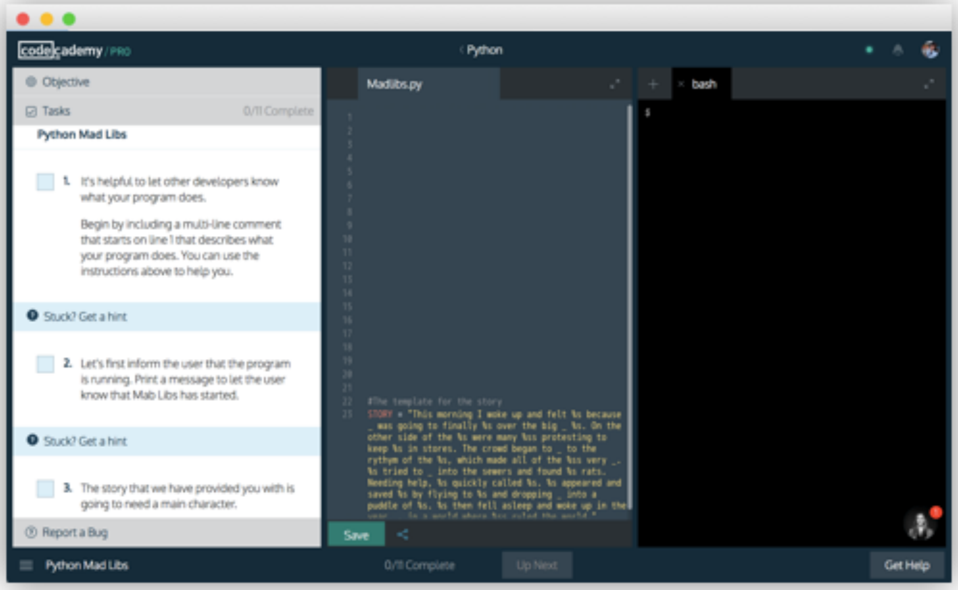
这门课程以实战为主，把一个涉及了爬虫、数据分析、网站开发的二手行情网站作为主项目，拆解成四个 level 的课程循序渐进。为了更好地分解练习，每节视频课程后面都穿插一个小项目。这门课提供了一条实战学习的最优路径，让你在最短时间掌握最关键的实战技能点。

发布了半年多，这门课已经成为网易云课堂上最畅销的 Python 课程，免费课程有4万多学员，付费课程有接近1千名学员。如果你想要快速上手实战项目，先掌握项目实践中最必要的技能，这会是适合你的选择。

目前有一个限时的读者优惠活动，点这里 https://www.wenjuan.com/s/JjU3uy/ 可以申请50元的课程优惠券。

2.CodeCademy

https://www.codecademy.com



CodeCademy 在增加了 pro 功能之后，变得更有竞争力了。付费每月20刀之后，在学习路径中除了知识点练习，还会阶段性地出现测试和真实项目。之前一直被诟病的缺少讲解环节，现在也通过更详细的文字讲解、论坛互动来解决了。

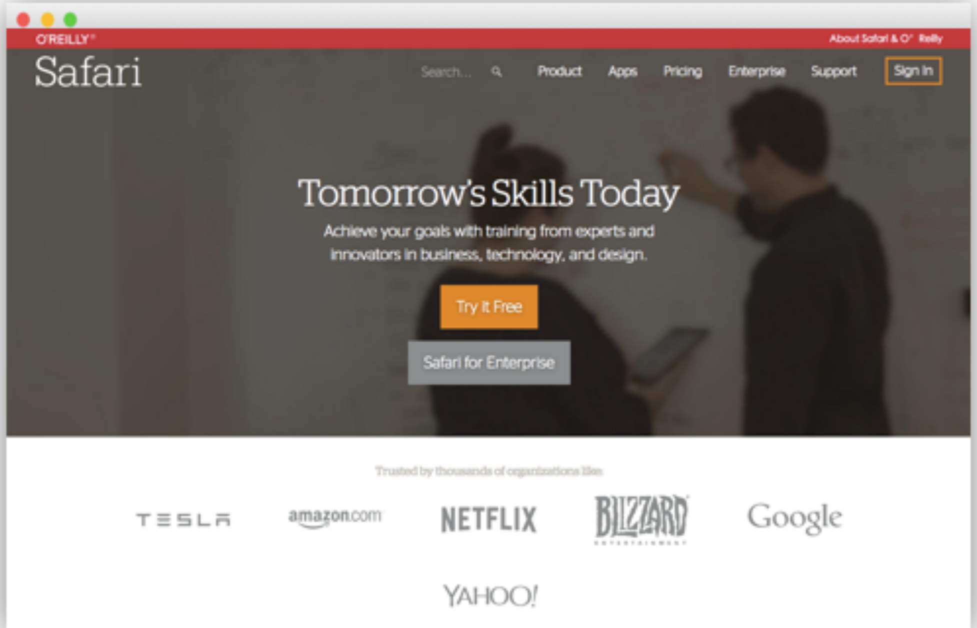
如果想要系统地把语法过一遍，是个不错的选择。不过由于课时拆解的太细，会出现一些 100*6% 之类的题目，可能会让你觉得无聊又漫长，导致难以坚持下来。

资料库参考

在实践的过程中，你会不断发现更多需要解决的问题，更多需要连接的未知，这时候到哪里去查阅资料呢?

1.safari online book

https://www.safaribooksonline.com



知乎上有人问，送程序员男友什么礼物好，其中一个答案就是 safari online。编程的英文书一般都很贵，但在这里只需要39刀的价格，就能包月看到几乎所有已经出版的专业开发书籍，甚至还有没出版的新书。很适合用来扫书，几乎你想要的答案在里面找到。

2.图灵社区

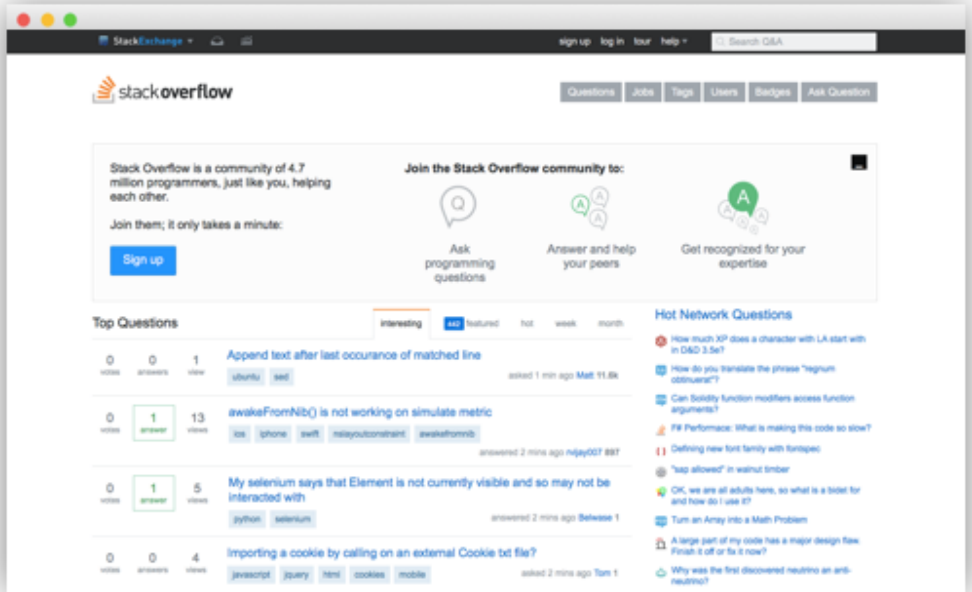
http://www.ituring.com.cn



技术方面的中文书籍，图灵出品，必属精品，我们这本书也是在图灵首发的。在图灵社区里面，有许多的免费或付费电子书，最新的翻译书籍也能在这里找到。还有一点我觉得和其他电子书平台很不一样，许多书会提供 PDF 版本，不是那种扫描的 PDF，而是和纸质书一样排版精良、易于阅读的 PDF 版本。由于技术书一般都是在电脑上看，这样会很方便。

3.Stackoverflow

http://stackoverflow.com



程序员的问答社区，试着把你的问题转换成一些关键词，里面总是会有非常好的回答，你会发现你踩过的坑总有人已经踩过了。

4.Python 官方文档

http://python.usyiyi.cn/python_343/tutorial/index.html



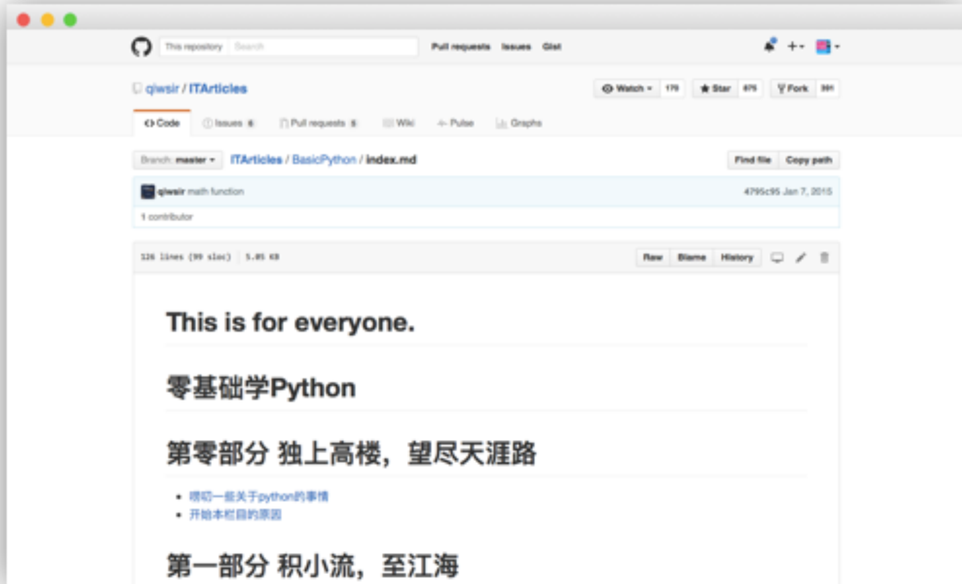
遇到需要较真的问题时，没有什么比 Python 官方文档说得更清楚的了。别对文档感到害怕，现在你已经熟悉一些 Python 基本常识了，只要有耐心就能读懂。

基础教程与书籍

这本书提供的是让读者从零到一入门，那么入门之后，读哪些书能进一步精深呢?

1.老齐的《零基础学 python》

https://www.gitbook.com/book/looly/python-basic/details



国内少见的不错的免费教程，适合补一下基础知识，缺点在于有点啰嗦，喜欢大段引用诗词、维基百科。

2.tutorialspoint 的中文仿站

http://www.runoob.com/python3/python3-tutorial.html



简单易懂的基础教程，同样适合补一下基础知识。

3.《Python 核心编程》

http://item.jd.com/11936238.html



这本书有众多“权威”和“精通”系列无法比拟的简洁和和精确，如果想深入细节，这本为最佳的选择。第3版比第2版增加了不少内容，务必选择第3版。

4.深入 Python

http://dipyzh.bitbucket.org



这是一本被低估甚至被诋毁的好书，事实上看不懂的部分你甚至可以当作科普读物来读都没有关系，因为它真得很有意思。

5.Real Python

https://realpython.com



鉴于国外的 python 生态环境比国内的要好上一大截，再加上作者是经验非常丰富的 Python 全栈开发者，所以以非常宽阔的视野写了这本由浅入深，非常实用的电子教程。作者还运营了 full stack python 这个网站。

6.python course

http://www.python-course.eu/python3_course.php



就职于 Saarland University 大学的计算机教授 Bernd Klein 所写的 python3 教程。别看网站有些简陋，这是我看过将近百余个 python 教程中集准确，通俗易懂，有趣众多优点于一身的优秀网络教程。

看完了

如果您对本书内容有疑问，可发邮件至 contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱: ebook@turingbook.com。

在这可以找到我们:

微博 @图灵教育 : 好书、活动每日播报

微博 @图灵社区 : 电子书和好文章的消息

微博 @图灵新知 : 图灵教育的科普小组

微信 图灵访谈 : ituring_interview，讲述码农精彩人生

微信 图灵教育 : turingbooks